UNIVERSITY OF CALIFORNIA
SANTA CRUZ

CONTROLLING NEURAL LANGUAGE MODELS FOR AFFECTIVE MUSIC COMPOSITION

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Lucas N. Ferreira

August 2021

The Dissertation of Lucas N. Ferreira
is approved:

Professor Jim Whitehead, Chair

Professor Marilyn Walker

Professor Levi Lelis

Peter F. Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Controlling Neural Language Models for

Affective Music Composition

by

Lucas N. Ferreira

Deep generative models are currently the leading method for algorithmic music composition. However, one of the major problems of this method consists of controlling the trained models to generate compositions with given characteristics. This dissertation explores how to control deep generative models to compose music with a target emotion. Given the limitation of labeled data, this dissertation focuses on search-based methods that use a music emotion classifier to steer the distribution of a pre-trained musical language model. Three different search-based approaches have been proposed. The first one is a genetic algorithm to optimize a language model towards a given sentiment. The second one is a decoding algorithm, called Stochastic Bi-Objective Beam Search (SBBS), which controls the language model at generation time. The third method is also a decoding algorithm but based on Monte Carlo Tree Search. SBBS has been applied to generate background music for tabletop roleplaying games, matching the emotion of the story being told by the players. A dataset of symbolic piano music called VGMIDI has been created to support the work in this dissertation. VGMIDI currently has 200 pieces labeled according to the circumplex model of emotion, and an additional 3,640 unlabelled pieces. The three methods were evaluated with listening tests, in which human subjects indicated that the methods could convey different target emotions.

To myself,

Lucas N. Ferreira,

the only person worthy of my company.

Acknowledgments

I want to thank ...

Chapter 1

Introduction

Music composers have been using algorithms, rules, and general frameworks for centuries as part of their creative process to compose music [101]. For example, Guido of Arezzo (around 991-1031), in his work *Micrologus*, described a system for the automatic conversion of text into melodic phrases. French composers of the *ars nova*, such as Phillipe de Vitry (1291–1361) and Guillaume de Machaut (1300–1377), used isorhythms as a method to map a rhythmic sequence (called *talea*) onto a pitch sequence (called *color*). In the *The Art of the Fugue*, Johann Sebastian Bach (1685–1750) deeply explored contrapuntal compositional techniques such as the fugue and the canon, both being highly procedural.

Since the 1950s, scientists, engineers, and musicians have been designing algorithms to create computer programs capable of composing music automatically. The *ILLIAC Suite* is considered the first piece to be fully composed automatically by an electronic computer [101]. Lejaren Hiller and Leonard Isaacson wrote the program that generated this composition with an ILLIAC computer at the University of Illinois [57]. Since then, many different methods

have been proposed to generate music with computers: expert systems [32], generative grammars [73], cellular automata [7], evolutionary algorithms [62], Markov chains [18], and neural networks [135]. The scientific (and artistic) field that organizes these algorithms is called *algorithmic music composition* (AMC). The work in this field has influenced music genres such as generative music [38] and supported applications in music analysis [85], procedural audio [40], audio synthesis [37], music therapy [142], among others.

With the great advances in deep learning since the 2000s, neural networks achieved impressive results in many areas of artificial intelligence (AI) such as computer vision, speech recognition, and natural language processing (NLP) [49]. Consequently, AMC researchers started exploring different types of neural networks to generate music: recurrent neural networks (RNNs) [105], transformers [63], convolutional neural networks (CNNs) [64], variational autoencoders (VAEs) [119], generative adversarial networks (GANs) [30], among others. Inspired by NLP, one of the most common approaches to neural AMC consists of using a transformer or a RNN to build a musical *language model* (LM).

1.1 Neural Language Models

In NLP, a LM is a joint probability function of sequences of tokens (e.g., words or characters) in a language [9]. Modern neural LMs compute the conditional probability of a token x_t given prefix tokens $\{x_1, x_2, \cdots, x_{t-1}\}$ by first computing a dense vector representation (embedding) of the prefix and then feeding it into a classifier to predict the next token [131]. Neural LMs can be trained from a text corpus and then used to generate new sentences similar

to the ones in the corpus. Typically, new sentences are generated in an autoregressive way. Namely, one starts with given prefix tokens $\{x_1, x_2, \cdots, x_{t-1}\}$ which are fed into the LM to generate the next token x_t. Next, x_t is concatenated with the prefix, and the process repeats until a special end-of-piece token is found or a given number of tokens are generated. Music can be seen as a sequence of musical tokens (e.g., notes, chords, and parts), and hence a musical LM can be defined to generate music similar to natural LMs. Such musical sequences are typically extracted from a corpus of symbolic music (e.g., MIDI or piano roll) [16]. Modern musical LMs have been capable of generating high quality pieces of different styles with strong short-term dependencies[1] [63].

Transformers and RNNs can learn a LM by processing input sequences of tokens $\{x_1, x_2, \cdots, x_{t-1}\}$ to predict, with a *softmax* activation function, an output distribution $\hat{y}_t$ for every token t. A *cross-entropy* loss function is then used to compare the predicted probability distribution $\hat{y}_t$, and the true next word y_t. RNNs process sequences step-by-step by keeping an internal state that is updated every step. Transformers process entire sequences in parallel, associating an *attention* score to each token, which determines how much that token contributes to the output of the network. Because transformers process tokens in parallel, they can take advantage of the parallel computing offered by GPUs, and hence can be trained considerably faster than RNNs [137]. One drawback of the transformers is that they can only process sentences with a fixed size instead of RNNs that can process sentences of any size.

[1] Supporting strong long-term dependencies (e.g., music form) is still an open problem.

1.2 Affective Algorithmic Composition

A major challenge of musical LMs consists of disentangling the trained models to generate compositions with given characteristics [42]. For example, one cannot directly control a LM trained on classical piano pieces to compose a tense piece for a horror scene of a movie. Being able to control the output of the models is especially important for the field of *affective algorithmic composition* (AAC), whose major goal is to automatically generate music that is perceived to have a specific emotion or to evoke emotions in listeners [141]. Applications involve generating soundtracks for movies and video games [140], sonification of biophysical data [21], and generating responsive music to support music therapy [94].

The AAC community has explored different ways to control AMC approaches. The traditional AAC methods are typically based on expert systems, evolutionary algorithms, and Markov chains. These methods require rules encoded by music experts to model principles from music theory to control the emotion of generated music. These methods are helpful in systematically investigating how a small combination of music features evoke emotions. However, due to the large space of features (e.g., tempo, melody, harmony, rhythm, timbre, and dynamics), it is challenging to create a fixed set of rules that consider all features. Data-driven methods (e.g., neural networks) do not have this problem because musical rules are learned directly from music data. The challenge with data-driven approaches is that it is relatively expensive to create datasets of music labeled according to a model of emotion. Thus, deep learning for AAC is still in its early days, and this dissertation is part of the first works in this area.

1.3 Contributions

This dissertation explores how to control neural LMs to generate music with a target emotion. Given the limitation of labeled data, the focus of this work is on search-based methods that use a music emotion classifier to steer the distribution of pre-trained musical LMs. With this framing, a high capacity LM L is pre-trained with a large unlabelled dataset and a music emotion classifier E is trained with the labeled data to predict emotions e. In order to boost the accuracy of the emotion classifier E, it is trained with transfer learning by fine-tuning the LM L with an additional classification layer. Three different search-based approaches have been proposed to control the LM L with the emotion classifier E to generate pieces with a target emotion e.

1.3.1 Learning to Generate Music with Sentiment

Inspired by the work of Radford et al. [113], the first explored approach is a genetic algorithm that optimizes the neurons of L that carry sentiment signal (positive or negative)[2], as given by E. A reasonably large labeled dataset called VGMIDI was created to train both L and E. All pieces in the dataset are piano arrangements of video game soundtracks. A custom web tool was designed to label these piano pieces according to the circumplex (valence-arousal) model of emotion [121]. Labeling music pieces according to emotion is a subjective task. Therefore, the pieces were annotated by 30 annotators via Amazon Mechanical Turk (MTurk), and the mean of these annotations was considered the ground truth. In this first work, the VGMIDI dataset had 95 labeled pieces and 728 unlabelled ones. The LM L was modeled as

[2] In this first work, only sentiment (and not emotions) was considered to simplify the problem.

a long short-term memory (LSTM) network pre-trained with these 728 unlabelled pieces. The sentiment classifier E was trained by fine-tuning L with an extra linear layer on the 95 labeled pieces.

L1 regularization was used while training E to enforce a sparse set of weights in E. This regularization highlighted the subset of neurons in L that carry sentiment signal. Thus, a genetic algorithm was used to optimize the weights of these L1 neurons to lead L to generate either positive or negative pieces. This approach was evaluated with a listening test where annotators labeled three pieces generated to be positive and three pieces generated to be negative. Results showed that the annotators agree that pieces generated to be positive are indeed positive. However, pieces generated to be negative are a little ambiguous, according to the annotators. This work was published in the Proceedings of the 20th Conference of the International Society for Music Information Retrieval (ISMIR19) [42].

1.3.2 Computer-Generated Music for Tabletop Role-Playing Games

The second approach is a variation of beam search, called *stochastic bi-objective beam search* (SBBS), to decode the outputs of L with the guidance of E into a sequence of musical tokens that convey e. Unlike the first approach, SBBS does not update the L weights to control L towards e. Instead, it steers the probability distribution of L in generation time by multiplying the probabilities of L with E. In this work, E is implemented as two independent binary classifiers: E_v for valence and E_a for arousal. At every decoding step, SBBS samples the next *beam* (set of candidate solutions) from this resulting distribution. SBBS applies *top k* filtering when expanding the search space in order to control the quality of the generated pieces.

In this work, the VGMIDI dataset was extended with extra 105 labeled pieces, increasing the number of labeled pieces to 200. Moreover, a new dataset of unlabelled piano pieces, called ADL Piano Midi, was created to train a larger L. ADL Piano Midi is composed of 11,086 piano pieces from different genres, where 9,021 of them were extracted from the Lakh MIDI dataset [116] and 2,065 were scraped from publicly available sources on the internet. The LM L was implemented with a GPT2 [115] transformer network pre-trained with the ADL Piano Midi. The emotion classifiers E_v and E_a were both trained by fine-tuning L with an extra linear layer on the 200 labeled pieces of the VGMIDI dataset.

SBBS was evaluated in the context of tabletop role-playing games. A system called *Bardo Composer* was built with SBBS to generate background music for game sessions of Dungeons & Dragons. Bardo Composer uses a speech recognition system to translate player speech into text, which is classified as having an emotion e. Bardo Composer then uses SBBS to generate musical pieces conveying the target emotion e. A user study showed that human subjects correctly identified the emotion of the generated music pieces as accurately as they were able to identify the emotion of pieces composed by humans. The contributions of this work were published in the Proceedings of the 13th [106] and 16th [41] AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE17 and AIIDE20).

1.3.3 Controlling Emotions in Symbolic Music Generation with MCTS

The third and most recent approach is another decoding algorithm that, similar to SBBS, does not update the L weights to control L towards the emotion e. This new decoding algorithm is based on *monte carlo tree search* (MCTS). At every step of the decoding process,

MCTS uses *predictor upper confidence for trees* (PUCT) to search over the space of sequences defined by L for solutions that maximize the average values of emotion given by E. MCTS samples from the distribution of node visits created during the search to decode the next token.

In this work, the VGMIDI dataset was extended with extra 2,912 unlabeled pieces, increasing the number of unlabeled pieces to 3,640. The LM L was implemented with a *music transformer* [63] pre-trained with these 3,640 unlabelled pieces. Unlike the second work, where E was split into two binary classifiers, E was trained as a single multiclass emotion classifier. Like the previous works, E was trained by fine-tuning L with an extra linear layer on the 200 labeled pieces of the VGMIDI dataset.

Two listening tests were performed to evaluate MCTS. The first one evaluates the quality of generated pieces, and the second one evaluates the MCTS accuracy in generating pieces with a given emotion. Results showed that MCTS is as good as SBBS in controlling emotions while improving music quality. An expressivity analysis of the generated pieces was also performed to show the music features being used to convey each emotion. The frequencies of pitch classes and note durations suggest that MCTS can reproduce some common composition practices used by human composers.

1.4 Dissertation Outline

This dissertation is organized as follows: Chapters 2 and 3 present the background work that this dissertation builds upon. While Chapter 2 presents an overview of algorithmic music composition, Chapter 3 dives into the fundamentals of deep learning for music genera-

tion. Chapter 4 reviews the previous methods to control the emotion of music composed algorithmically. It also reviews techniques developed within the NLP community to control LMs for different text generation tasks. Chapter 5 describes the work published at ISMIR19, where a genetic algorithm is used to fine-tune a pre-trained LSTM. Chapter 6 presents the work published at AIIDE20, which uses SBBS within Bardo Composer to compose music for tabletop roleplaying games. Chapter 7 presents this dissertation's most recent contribution: an MCTS decoding algorithm to control LMs to generate music with a target emotion. Chapter 8 discusses the weakness of the methods proposed in this dissertation in order to highlight different directions of future work. Finally, Chapter 9 concludes this dissertation.

Chapter 2

A Brief History of Algorithmic Music Composition

Algorithmic music composition (AMC) can be literally defined as the use of algorithms to compose music. This broad term includes non computational procedures from music theory developed to guide music composition and computational methods designed to generate music automatically or semi-automatically. This chapter presents a brief history of AMC to contextualize this dissertation, from the first procedures created by medieval music theorists to the modern computational methods designed by scientists and engineers. Most computational methods are briefly discussed with a few examples in this chapter, except neural networks, which are covered in greater detail in the next chapter. Moreover, it is important to highlight that this dissertation focuses on symbolic music composition, and hence audio-based methods are not covered in this chapter.

2.1 Procedures in Music Theory

Music composers have been developing procedures for centuries to compose different aspects of music pieces. Guido of Arezzo (around 991-1031) developed one of the earlier examples of AMC in his work *Micrologus*, where he described a method for mapping Latin lyrics into melodies. The method extracts the vowels from given lyrics and then maps the vowels into pitches. Figure 2.1 shows how Guido of Arezzo mapped the Latin vowels into the seven different natural pitches of the western music system[1].

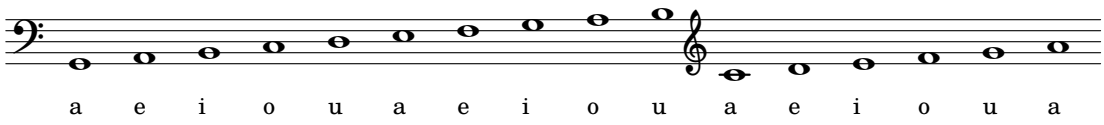


Figure 2.1: Mapping of vowels on natural pitches by Guido of Arezzo.

Since there are more pitches than vowels, some vowels are mapped to different pitches. When mapping vowels that have more than one associated pitch, the pitch should be selected at random. For example, consider the verse from the *Hymn to St John* in Figure 2.2, in which the vowels are shown in parenthesis. With the mapping rules defined in Figure 2.1, the first line of the verse can be mapped to the melody *DBFCCE*.

Around 1280, the music theorist Franco de Cologne, in his work *Cantus Mensurabilis*, introduced a music notation system where the note durations are defined by their shapes. This new notation allowed composers to treat rhythm independently from pitch. For example, French composers of the *ars nova*, such as Phillipe de Vitry and Guillaume de Machaut, used

[1] C, D, E, F, G, A, B

REsonare fibris (eoaeii)

MIra gestorum (iaeou)

FAmuli tuorum (auiuou)

SOLve polluti (oeoui)

LAbii reatum (aiieau)

Sancte Ioannes (aeioae)

Figure 2.2: A verse from the *Hymn to St John*.

a technique called isorhythm to map a rhythmic pattern (named the *talea*) onto a pitch contour (named the *color*). Figure 2.3 shows the tenor melody of the piece *De bon espoir-Puisque la douce-Speravi* by Guillaume de Machaut built using the isorhythm technique.

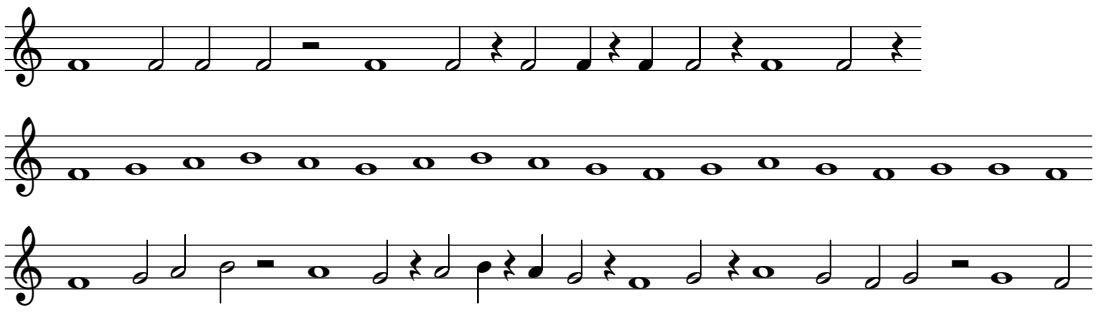


Figure 2.3: The tenor melody of the piece *De bon espoir-Puisque la douce-Speravi* by Guillaume de Machaut.

The first staff is the talea and the second one is the color. The third staff is the resulting melody of applying the former onto the latter. In this example, the talea has twelve notes and five rests, whereas the color has eighteen notes. The note durations of the talea are applied in order onto the respective notes of the color. Whenever there is a rest in the talea, that rest is

merged into the pitches of the color. If the last note of the talea is mapped and there are still notes left in the color, the mapping process continues from the first note of the talea.

In the Baroque Period (1600–1750), Johann Sebastian Bach wrote *The Art of Fugue*, a musical work that explores different *fugues* and *canons* from a single musical subject[2]. Fugues and canons are highly procedural contrapuntal compositional techniques. Counterpoint consists of interleaving two or more melodies that are harmonically dependent but independent in rhythm and melodic contour. For example, in a canon, the composer starts with a melody, called the *leader*, which is strictly followed at a delayed time interval by another voice, called the *follower*. The follower may present a variation of the leader through transformation operations such as transposition[3], augmentation[4], or inversion[5] [126]. For example, Figure 2.4 shows the first 13 measures[6] of the *Canon per Augmentationem in Contrario Motu* from *The Art of Fugue*.

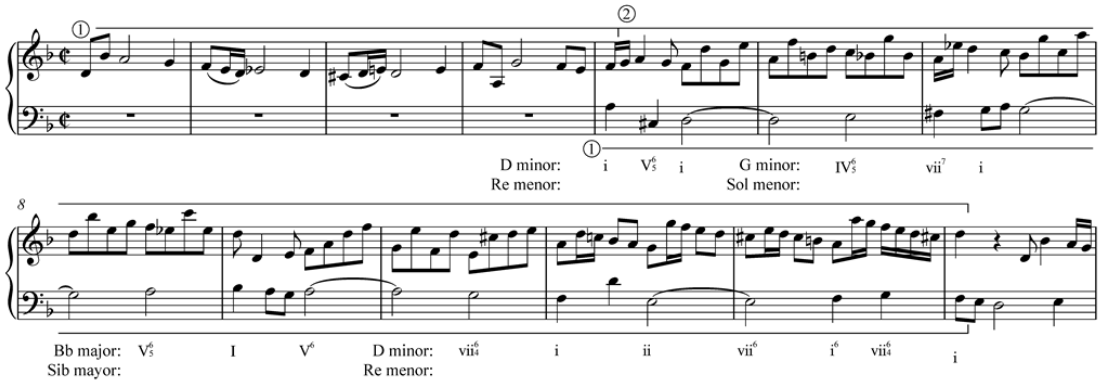


Figure 2.4: The first 13 measures of the *Canon per Augmentationem in Contrario Motu* from *The Art of the Fugue* by Johann Sebastian Bach.

[2]A subject is the material, usually a recognizable melody, upon which part or all of a composition is based.

[3]Moving a set of notes up or down in pitch by a constant interval

[4]Repeating a set of notes with longer durations.

[5]Playing a given set of notes upside down, reversing the contour of the notes.

[6]A measure (or bar) refers to a single unit of time featuring a specific number of beats played at a particular tempo. Measures are indicated by vertical bar lines on the staff.

The first 4 measures of the upper voice present the subject (leader) of the canon. In measures 5 to 13, the lower voice (follower) transforms the leader using augmentation and inversion. Note that each ascending interval in the leader becomes a descending interval in the follower. Figure 2.5 highlights these transformations by aligning the leader (upper voice) with the follower (lower voice). Due to augmentation, the follower needs 8 measures to answer the first 4 measures of the leader.

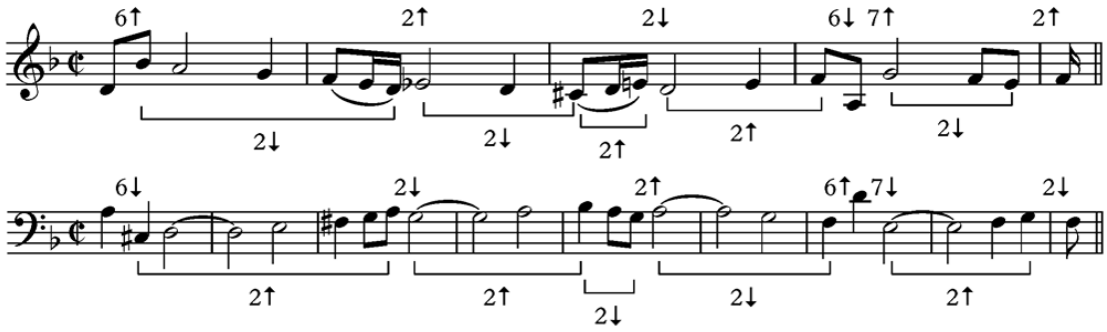


Figure 2.5: The leader (upper voice) aligned with the follower (lower voice) of the *Canon per Augmentationem in Contrario Motu* from *The Art of the Fugue*.

In the Classical Period (1750-1827), *Musikalisches Würfelspiel* (german for a musical dice game) became a popular method to generate music randomly. It consists of selecting precomposed snippets of music according to the result of dice rolls. One of the most famous applications of this method is attributed to Wolfgang Amadeus Mozart, although this attribution has not been authenticated [22]. Mozart's dice game was designed to generate sixteen-measure-long minuets[7]. The game works by creating an eleven-by-sixteen table, where the rows represent possible results of rolling two six-sided dice and columns are the indices of each measure of the minuet. Each element in the table is a precomposed measure.

[7] A minuet is a classic form of dance from the classical period.

	Part 1							
	I	II	III	IV	V	VI	VII	VII
2	96	22	141	41	105	122	11	30
3	32	6	128	63	146	46	134	81
4	69	95	158	13	153	55	110	24
5	40	17	113	85	161	2	159	100
6	148	74	163	45	80	97	36	107
7	104	157	27	167	154	68	118	91
8	152	60	171	53	99	133	21	127
9	119	84	114	50	140	86	169	94
10	98	142	42	156	75	129	62	123
11	3	87	165	61	135	47	147	33
12	54	130	10	103	28	37	106	5

Table 2.1: Example of Mozart's dice game. Each element in the table is an integer representing the number of a precomposed measure.

Table 2.1 shows an example of Mozart's dice game with only the first part of the minuet and hence only eight columns. To generate the first part of the minuet with this implementation, one has to roll two six-sided dice for each column j of Table 2.1. After each roll, the sum i of the two dice is used to look up the row number i for column j. The element i, j in Table 2.1 is then used to retrieve a single measure from a collection of musical fragments. Figure 2.6 shows the first eight measures of a minuet that can be generated using Table 2.1.

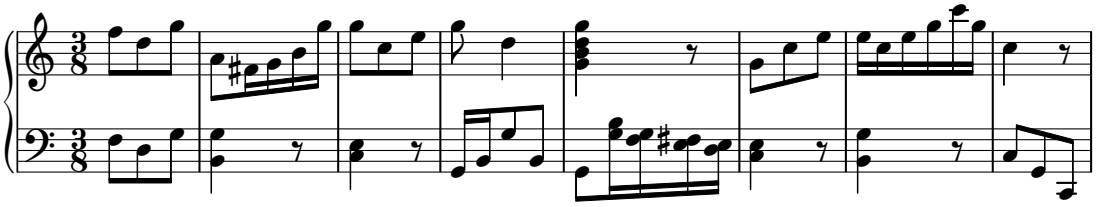


Figure 2.6: Example of minuet generated using Mozart's dice game.

In the Romantic Period (1800–1850), composers developed a harmonic vocabulary

with extensive use of chromaticism[8]. In the transition from Romanticism to Modernism, Arnold Schoenberg, with his students, Anton Webern and Alban Berg, established new procedures for music composition called *Twelve-tone serialism*. Serial composition consists of arranging a series of musical elements (e.g., pitches and rhythms) into a pattern that repeats itself throughout a composition. A basic form of serial composition consists of selecting a given number of notes on the chromatic scale[9] and creating permutations using only that number of notes. The selected notes, called the *row*, must all be played once before repeating, although a note can be repeated immediately after it has been played (for example, A, and then A). The first arrangement of the row is called the *tone row*. Transformations such as transposition, inversion, retrograde[10], or retrograde inversion[11] can be applied to the tone row to introduce variation into a serial composition.

Twelve-tone serialism is a serial technique where the tone row is an ordered arrangement of all the twelve notes of the chromatic scale. The goal of this technique was to replace tonal music, which is built based on keys (such as C major or D minor). By focusing on the twelve notes of the chromatic scale, no emphasis is given to any single key. Figure 2.7 illustrates the tone row used by Alban Berg in the *Lyric Suite* for string quartet composed in 1926.

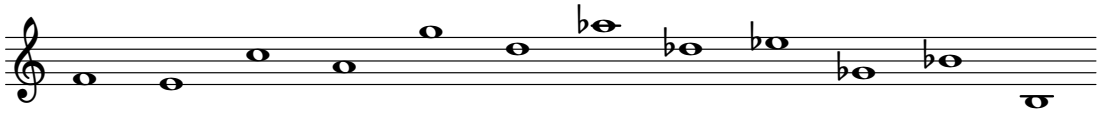


Figure 2.7: Tone row used by Alban Berg in the *Lyric Suite* for string quartet composed in 1926.

[8]Chromaticism is the use of notes outside the scale of which a composition is based.
[9]A musical scale with twelve pitches, each a semitone, above or below its adjacent pitches.
[10]Playing a sequence of notes backwards.
[11]Playing a sequence of notes backwards and upside down.

Table 2.2 shows different transformations that can be applied to the tone row showed in Figure 2.7. The first row, when read from left to right, is the tone row. When read from right to left, the first row is the retrograde form of the tone row. The inversion of the tone row is the first column when read from top to bottom. The retrograde inversion is found by reading the first column from bottom to top. Each row and column is labeled with T_i, where $0 \leq i \leq 11$. A label T_i means that the respective row is the transposition of the tone row by i half steps. For example, T5 means the tone row has been transposed up five half steps.

	T0	T11	T7	T4	T2	T9	T3	T8	T10	T1	T5	T6
T0	F	E	C	A	G	D	Ab	Db	Eb	Gb	Bb	B
T1	Gb	F	Db	Bb	Ab	Eb	A	D	E	G	B	C
T5	Bb	A	F	D	C	G	Db	Gb	Ab	B	Eb	E
T8	Db	C	Ab	F	Eb	Bb	E	A	B	D	Gb	G
T10	Eb	D	Bb	G	F	C	Gb	B	Db	E	Ab	A
T3	Ab	G	Eb	C	Bb	F	B	E	Gb	A	Db	D
T9	D	Db	A	Gb	E	B	F	Bb	C	Eb	G	Ab
T4	A	Ab	E	Db	B	Gb	C	F	G	Bb	D	Eb
T2	G	Gb	D	B	A	E	Bb	Eb	F	Ab	C	Db
T11	E	Eb	B	Ab	Gb	Db	G	C	D	F	A	Bb
T7	C	B	G	E	D	A	Eb	Ab	Bb	Db	F	Gb
T6	B	Bb	Gb	Eb	Db	Ab	D	G	A	C	R	F

Table 2.2: Different transformations that can be applied to the tone row showed in Figure 2.7.

In the 20th century, Iannis Xenakis deeply explored the use of statistical methods to compose *stochastic music* [145] – music in which some elements of the composition are defined randomly. For example, in *Pithoprakta* (1956), he used Gaussian distributions to define the "temperatures" of massed glissandi[12]. In *Achorripsis* (1957), he used Poisson's distribution of rare events to organize "clouds" of sound [2]. John Cage is another important composer of

[12]Continuous transition between two notes of different pitches.

the 20th century who worked with stochastic music. Cage's *Music of Changes* is a piece for solo piano which was composed using the *I Ching*, a Chinese classic text that is commonly used as a divination system. The I Ching was applied to randomly generate charts of pitches, durations, dynamics, tempo, and densities.

2.2 Computational Methods

The *ILLIAC Suite: String Quarter No 4* [57], a composition for string quartet, is considered to be the first music piece to be entirely generated by a digital computer. This piece was generated in 1956 by an ILLIAC computer programmed by Lejaren Hiller and Leonard Isaacson at the University of Illinois. The *ILLIAC Suite* has four movements with melodies that increase in complexity across the movements. The first movement used counterpoint rules from the Renaissance to generate simple polyphonic melodies. The second movement used a random chromatic method that explored aesthetic differences between seventeenth and twentieth-century musical styles. For the third and fourth movements, Hiller and Isaacson manually designed a Markov chain to generate melodies with the style of Arnold Schoenberg's twelve-tone music.

Xenakis and Cage were also pioneers in the use of computer algorithms to compose music. In 1962, Xenakis wrote the *Stochastic Music Program* in the FORTRAN programming language, which employed probability functions to determine the global structure (e.g., length of sections and density) and the note parameters (e.g., pitch and duration) of his compositions [89]. *Morsima-Amorsima* is an example of piece composed by Xenakis with the support of this program. From 1967 to 1969, John Cage partnered with Lejaren Hiller to compose a mul-

timedia piece called *HPSCHD*, which used a dice game approach with precomposed snippets from pieces by Mozart, Beethoven, Chopin, and others. There are several other early examples of computational approaches to AMC, such as the *Push Button Bertha* [2], a piece generated in 1956 by a DATATRON computer programmed by Martin Klein and Douglas Bolitho at the company Burroughs, Inc. Another important early example is the PROJECT1 (1964), a computer program written by Gottfried Michael Koenig that used serial composition and Markov chains to compose pieces such as the *Project 1, Version 1* for 14 instruments.

Most of these early computational examples of AMC were developed by artists in an *ad hoc* way. More recently, computer scientists and engineers started to explore AMC more systematically and a wide range of methods have been proposed: expert systems [46], generative grammars [22], cellular automata [95], evolutionary algorithms [62], Markov chains [56], neural networks [135], and others. The remainder of this chapter briefly introduces these methods, except neural networks, which are discussed in greater detail in the next chapter.

2.2.1 Expert Systems

AMC *expert systems* use rules that manipulate symbolic music to mimic the reasoning of music composers. For example, Gill [46] presented the first application of hierarchical search with backtracking to guide a set of compositional rules from Schoenberg's twelve-tone technique. Many different works formulated AMC expert systems as a *constraint satisfaction problem* (CSP) [4]. For example, Ebcioğlu [32] designed a system called CHORAL for harmonizing four-part chorales in the style of J.S. Bach. The system contains over 270 rules (related to melody, harmony, etc.), expressed in the form of first-order predicate calculus. CHORAL

harmonizes the chorales using an informed search method where the heuristics guide the search towards Bachian cadences[13]. AMC expert systems can also be formalized with *case-based reasoning*. For example, Pereira et al. [110] proposed a system with a case database from just three Baroque music pieces, which were analyzed into hierarchical structures. The system composes just the soprano melodic line of the piece by searching for similar cases in its case database.

2.2.2 Generative Grammars

Generative grammars are a set of expansion rules that give instructions for how to expand symbols from a vocabulary. Starting with an initial sequence of symbols, one can compose music with generative grammars by recursively applying expansion rules until a terminal symbol has been reached or a desired length of music has been generated [61]. The expansion rules can be manually defined or inferred from analyzing a corpus of pre-existing music compositions. Lidov and Gadura [87] presented an early example of a generative grammar manually designed for the generation of melodies with different rhythmic patterns. A more recent example is the work of Keller and Morrison [73], who designed a probabilistic generative grammar for the automatic generation of convincing jazz melodies. In this case, the expansion rules have probabilities associated with them.

One of the most famous examples of generative grammars in AMC is Cope's *Experiments in Musical Intelligence* (EMI) [22], which automatically derives a special type of grammar called an *augmented transition network* from a corpus of compositions in a specific style. EMI extracts this augmented transition network by finding short musical patterns that

[13] A cadence is a chord progression that occurs at the end of a phrase. A phrase is a series of notes that sound complete even when played apart from the main song.

are characteristic of the style being analyzed. EMI also determines how and when to use these patterns in compositions with that style. The inferred transition network can be used to generate new music pieces with the style of the analyzed corpus.

2.2.3 Cellular Automata

Cellular Automata (CA) is a dynamic system composed of simple units (called *cells*) usually arranged in an n-dimensional grid. A cell can be in one state at a time. At each time step, the CA updates each cell according to *transition rules*, which consider the state of the cell and/or the state of the neighbor cells [143]. In his 1986 work *Horos*, Xenakis designed a CA to produce harmonic progressions and new instrument combinations [130]. CAMUS [95] is a system that combined two bi-dimensional CAs to compose polyphonic music: Conway's *Game of Life* [45] and Griffeaths *Crystalline Growths* [26]. Each activated cell in the Game of Life was mapped to a triad[14], whose instrument was selected according to the corresponding cell in the Crystalline Growths CA. WolframTones [7] is a commercial system that composes music with one-dimensional CAs that resemble a selected musical style. The system allows users to select a pre-defined music style (e.g., classical, ambient, and jazz) and set different parameters of the algorithm (e.g., rule number, rule type, and seed). Moreover, users can define how to map the CA patterns to different musical features (e.g., pitch, tempo, and timber).

[14] A tuple of three notes.

2.2.4 Evolutionary Algorithms

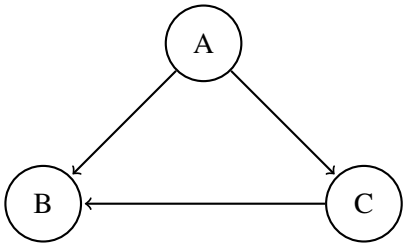
Evolutionary Algorithms (EAs) are optimization algorithms that keep a population of candidate solutions (called *individuals*) to maximize (or minimize) a given objective function (called *fitness function*) with an iterative process: (a) evaluation of the current population with the given fitness function, (b) selection of the best solutions and (c) generation of new solutions from the selected solutions. Horner and Goldberg [62] presented one of the first examples of EA for music composition. Their algorithm was inspired by a composition technique called *thematic bridging*, where the beginning and end of a piece are given, and a fixed number of transformations (e.g., transposition, inversion, and retrograde) are applied to map the beginning into the ending. Each candidate solution was encoded as a fixed set of transformations. The fitness function measured the distance between the ending generated by the candidate and the given ending. New solutions are generated with regular mutation and 1-point crossover. Other examples of EAs include the works of McIntyre [92] for four-part baroque harmonization, Polito et al. [111] for counterpoint composition, and Papadopoulos et al. [108] for the generation of melodies for given jazz chord progressions.

Formulating good fitness functions is one of the major challenges of applying evolutionary algorithms for AMC. To deal with this problem, a wide range of works use human evaluators to listen and judge the fitness of the candidate solutions. These approaches are called *interactive genetic algorithms* (IGAs). For example, GenJam [13] is a IGA for generating jazz solos with two hierarchically structured populations: one for bar units and the other for jazz phrases (constructed as sequences of measures). A human evaluator defines the fitness of the

candidate solutions with a binary score (good or bad). GenJam accumulates these scores to select phrases during the evolutionary process and generate the solos for a given chord progression. Other examples of IGAs are presented in the works of Jacob [66], Schmidl [122], and Tokui et al. [136].

2.2.5 Markov Chains

Markov chains were a very popular method in the early days of AMC [3]. A Markov chain is a system with a sequence of states, using conditional probabilities to model the transitions between successive states [47]. In a first-order Markov chain, the probability of the next state depends only on the current state, but in an nth-order Markov chain, the probability is conditioned on the previous n-1 states [47]. Markov Chains can be represented as a directed graph where nodes represent states, edges represent transitions between states, and edge weights represent the probability transition between states. This graph can be mapped to a probability table T where rows i and columns j represent nodes, and elements $T_{i,j}$ represent the probability of transitioning between nodes i and j. Figure 2.8 shows an example of a simple abstract Markov Chain.



	A	B	C
A	0.0	0.1	0.9
B	0.1	0.0	0.9
C	0.5	0.5	0.0

Figure 2.8: Example of Markov Chain represented as a directed graph with its respective probability table.

Markov chains can be derived manually with the support of music theory or learned from a corpus of music pieces. In both cases, one has to define how to encode music symbols into a sequence of states [3]. To learn a Markov chain from a corpus, one can count, for each state s_1 in the corpus, the number of times s_1 appears after each other state s_2. The transition probability table can then be constructed by normalizing these counts with the total number of transitions in the corpus. The third and fourth movements of the *ILLIAC Suite* are the earliest examples of manually-designed Markov chains, while Brooks et al. [18] presented one of the first examples inferred from a corpus of music. Brooks et al. [18] experimented with different orders of Markov chains where each state represents a pitch class. The probability tables of these chains were learned from 37 common meter ($\frac{4}{4}$) hymn tunes (monophonic). While Markov chains designed manually by composers worked well for specific compositional tasks [134, 70, 81], those learned from a corpus, in practice, can capture only short-term dependencies in music [99]. Moreover, low order chains typically generate unmusical compositions that wander aimlessly, while high order ones tend to repeat segments from the corpus and are very expensive to train [99].

Markov chains can also be used to evaluate music generated by other methods. Given a sequence of states representing a piece of music, one can evaluate this piece with the joint probability of the sequence as given by the Markov chain. Thus, the higher the probability, the better the music. This approach was used by Lo and Lucas [88]. They trained a Markov chain with classical music pieces and used it to calculate the fitness of candidate solutions of an EA. Each solution in the EA represents a melody encoded as a sequence of pitch numbers as defined by the MIDI format.

Most methods presented in this section require manually designed musical models (or rules) to manipulate or evaluate music. Although music theory formalizes several aspects of music analysis and composition, designing computational music models for AMC is still very challenging, given that music composition is a task that involves human creativity. Some of the presented methods, such as generative grammars and Markov models, can infer composition rules from a music corpus. However, in practice, they have limited performance in terms of music quality. Deep learning is a modern approach to AI, where neural networks are trained to perform various tasks. Recently, the AI research community has drawn substantial attention to this approach due to the impressive results that it has been achieving in different problems (e.g., image classification, speech recognition, and machine translation) [84]. These results also motivated AMC researchers to explore deep learning algorithms for music composition [16]. The next chapter presents a detailed discussion on how (deep) neural networks can learn music models from a symbolic music corpus.

Chapter 3

Deep Learning for Algorithmic Music Composition

Deep Learning is a class of *Machine Learning* (ML) algorithms based on *Neural Networks* (NNs) with multiple layers that progressively extract higher-level features from raw data (e.g., text, images, audio, and video) [49]. As ML algorithms, deep neural networks are used to learn different tasks from examples without being explicitly programmed to do so, including *supervised learning*, *unsupervised learning*, and *reinforcement learning* tasks. In a *supervised learning* task, pairs (X,Y) of inputs X and target classes (also called labels) Y are provided by a dataset as training examples. The NN then is *trained* to learn a function that maps the input examples into the target classes. The learned function is typically used to perform predictions (e.g., classification) on examples that the NN has not seen during training. Supervised learning is typically divided into *binary* and *multiclass* problems. In binary problems, a given input x can have one of two possible values $y \in [0,1]$. In multiclass problems, the label $y \in [0,1,\cdots,L]$ can

take one of $L > 2$ values. Classic examples of binary and multiclass problems are email spam detection (spam or not) and handwritten digits classification [83], respectively. In unsupervised learning tasks, only the inputs X are given in the dataset, and the NN is trained to learn internal patterns in the data. These learned patterns can be used for different purposes such as clustering, transfer learning, and generative modeling [10]. In reinforcement learning, the NN is trained to learn an agent that can take optimal actions in an environment according to a *reward function*.

Modern neural AMC systems are typically designed in an unsupervised learning setting, where a NN has to learn relationships between different music structures (e.g., notes, chords, and melodies) represented in symbolic format. Formally, these NNs are *generative models*, i.e. a model that captures a probability distribution $P(X)$ from a given dataset X. Inspired by the great results that deep learning has achieved in NLP, generative models for AMC are typically designed as neural *language models* (LM). In NLP, a LM is a conditional probability $L = P(x_t|x_1, x_2, \cdots, x_{t-1})$ of the next token x_t given a prefix with the $t-1$ previous tokens $\{x_1, x_2, \cdots, x_{t-1}\}$ of a sentence. One can train a NN to learn L by processing input sequences of tokens $\{x_1, x_2, \cdots, x_{t-1}\}$ to predict the next token x_{t+1} from the current token x_t. A NN trained this way can generate new sentences by sampling tokens from L or searching for sequences over the space defined by L.

Considering that music is a sequence of musical tokens (e.g., notes, chords, and sections), one can train a neural LM L to compose music by (a) creating a dataset of symbolic music, (b) designing a NN to learn L, and (c) sampling or searching tokens with L. The remainder of this chapter discusses different approaches for these three steps.

3.1 Symbolic Music Representation

Symbolic music representation refers to using high-level symbols such as tokens, events, or matrices as a representation for music modeling. The advantage of symbolic music representation over audio music representation is that the former incorporates higher-level features (e.g., structure, harmony, and rhythm) directly within the representation itself, without the need for further preprocessing. There are many formats to represent symbolic music in computers, but the most common ones are MIDI and piano roll.

3.1.1 MIDI

MIDI is a standard protocol for interoperability between various electronic instruments, devices, and software [16]. A MIDI file represents a music piece as a series of messages that specify real-time note performance data and control data. The two most important MIDI messages for music LMs are the following:

- NOTE_ON: this message is sent when a note starts, and it has three parameters:
  - *Channel number*: indicates the instrument track with an integer $0 \leq i \leq 15$
  - *Note number*: indicates the note pitch with an integer $0 \leq p \leq 127$
  - *Note velocity*: indicates how loud the note is played with an integer $0 \leq v \leq 127$
- NOTE_OFF: this message is sent when a note ends, and it has the same three parameters as the NOTE_ON message. In this case, the velocity parameter indicates how fast the note is released.

Note events are organized into a stream format called *track chunk*, which specifies the timing information of each note event with a delta time value. A delta time value represents the time of the note event either in relative metrical time (number of ticks from the beginning) or absolute time. In the relative metrical format, a reference called *division* is defined in the file header to set the number of ticks per quarter note. Table 3.1 shows an example of a MIDI *track chunk* encoded in a readable format, where the time division has been set to 384 ticks per quarter note.

Delta time	**Event Type**	**Channel**	**Pitch**	**Velocity**
96	NOTE_ON	0	60	90
192	NOTE_OFF	0	60	0
192	NOTE_ON	0	62	90
288	NOTE_OFF	0	62	0
288	NOTE_ON	0	64	90
384	NOTE_OFF	0	64	0

Table 3.1: Example of MIDI file encoded in a readable format [16].

3.1.2 Piano Roll

Piano roll is another common format of symbolic music. It is inspired by classic automated pianos that play pieces without a human performer by reading music from a continuous roll of paper with perforations punched into it. Each perforation automatically triggers a note, where the perforation location defines the note pitch, and the perforation length defines the note duration. In a modern piano roll, music is divided into discrete time steps forming a grid where the x axis represents time and the y axis represents pitch. The values $0 \leq v \leq 127$ in the grid represent the velocity of the notes. Figure 3.1 shows an example of a modern piano roll.

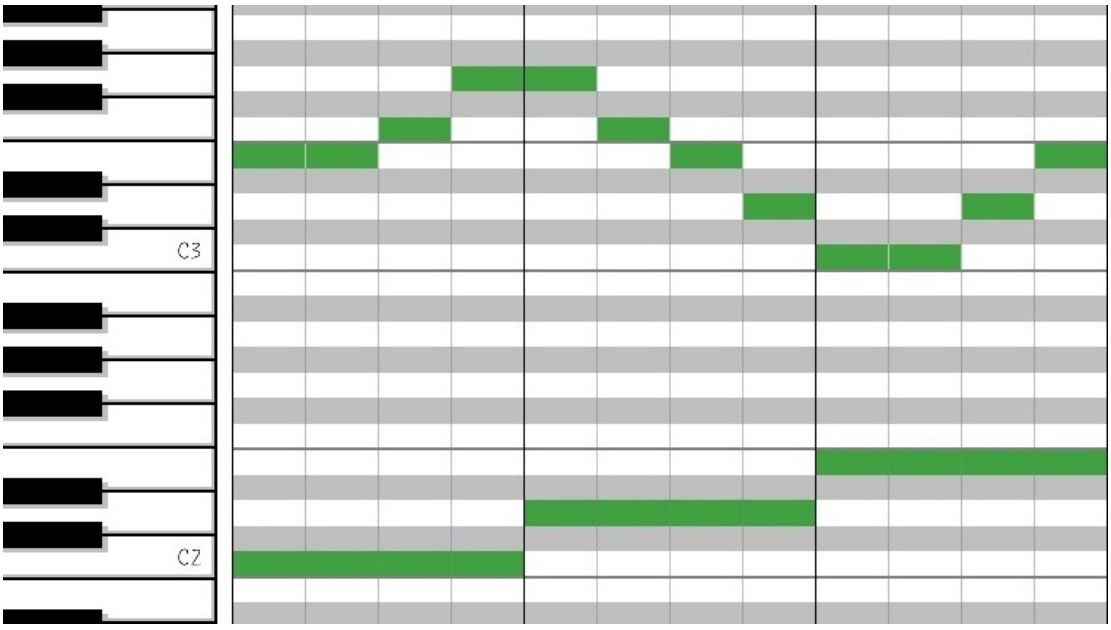


Figure 3.1: Example of music represented in the piano roll format.

The MIDI representation can be mapped to a piano roll by sampling time steps with a given frequency (e.g., every 16th note) from the MIDI events. Because of this property, most datasets of symbolic music organize pieces in a collection of MIDI files. For example, the *MAESTRO* dataset [53] is composed of about 200 hours of virtuosic piano performances of classical music pieces captured from the International Piano-e-Competition [1] in MIDI format aligned with audio waveforms. The *Lakh* [116] dataset is a collection of 176,581 unique MIDI files from various music genres (mostly pop music) scraped from publicly available sources on the internet, where 45,129 of them have been matched and aligned to entries in the *Million Song Dataset* [12]. *Piano midi.de* is a dataset of classical piano pieces from a wide variety of composers recorded in MIDI with a digital piano. JSB Chorales [14] contains the entire corpus of 382 four-part harmonized chorales by J. S. Bach.

MIDI and piano roll are the most common formats used to represent symbolic music. However, other formats have also been used in the AMC literature. For example, the ABC notation [139] is a text-based music notation system popular for transcribing, publishing, and sharing folk music. MusicXML [48] is a markup language that has been designed to facilitate the sharing, exchange, and storage of scores by musical software systems.

To use any of these symbolic music representations with LMs, one has to define a vocabulary that encodes music data into a sequence of music symbols. For example, in a MIDI representation, one has to map the note events into tokens and use the delta-time information from the *track chunks* to define the order of the tokens. In a piano roll representation, one has to map the vertical axis (pitch) into tokens and process the piano roll grid either horizontally or vertically to define the order of the tokens. To be processed by NNs, each token in the vocabulary has to be mapped into a vector. Traditionally, these tokens are mapped using *one-hot* encoding, where each token is given an index i and is represented by a vector $v = [v_1, v_2, \cdots, v_n]$, where only $v_i = 1$ and all the other dimensions $v_{j \neq i} = 0$. In the one-hot encoding, n is the number of tokens in the vocabulary. For example, considering a vocabulary $V = \{a, b, c, d, e\}$, the one-hot encoding of the token c is $c = [0, 0, 1, 0, 0]$.

3.2 Neural Networks

Artificial Neural Networks, or simply Neural Networks (NNs), interconnect a number of simple processing units called *neurons* to learn a function from training examples. These neurons are typically organized into layers. Neurons might be connected to several other neu-

rons in the layer before it, from which it receives data, and several neurons in the layer after it, to which it sends data. NNs can be defined with different *architectures*, i.e. with a different number of layers (*depth*) and different layouts of neuron connections. The first layer of the network is called the *input layer*, and the last one is called the *output layer*. All the intermediate layers are called *hidden layers*. Each neuron in the hidden or output layers takes as input a vector x of incoming connections from the previous layer and assigns a weight vector w to these connections. In its most basic form, the neuron first applies a linear transformation $z = w\dot{x} + b$ to the inputs x, where b is an extra weight called *bias* that is not tied to any neuron of the previous layer. The neuron then uses a nonlinear function called *activation function* f to map the linearly transformed inputs z into an output $\hat{y}$. Figure 3.2 shows a three-layer[1] NN called *feedforward* network or *multilayer perceptron* (MLP).

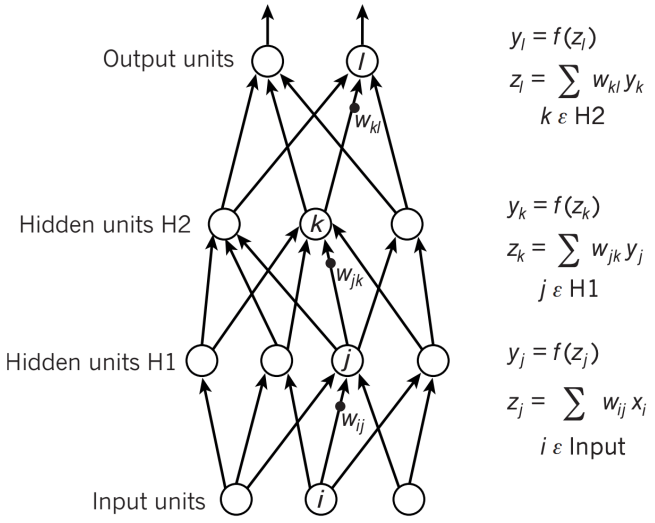


Figure 3.2: Diagram of a feedforward neural network [84]. The computations performed by the neurons l (hidden layer H1), k (hidden layer H2), and l (output layer) are highlighted beside them. The bias terms have been removed for clarity.

[1]Typically, the input layer is not considered when counting the depth of the NN.

The number of neurons per layer (i.e., the *layer size*) and the layers' activation functions depend on the task being learned by the NN. In traditional supervised problems, the input layer size is defined by how the input examples are represented and the output layer size by the number of classes in the problem. For example, consider a handwritten digits classification problem in which each handwritten digit is stored in a 28x28 grayscale image. The goal is to classify the images into one of the ten digits (0 to 9). In this example, the input layer size is 784 neurons, one for each pixel in the image. The output layer size is 10 neurons, one for each class. The sizes of the hidden layers are defined arbitrarily and should be controlled to optimize the performance (e.g., classification accuracy) of the network.

In *multiclass* problems, such as the handwritten digit classification, the *softmax* activation function is used in the output layer to create a probability distribution over the classes. Thus, the NN predicts the class with maximum probability. In *binary* problems, the *logistic* activation function is typically used to map the output layer into the probability that the label is one $P(y = 1)$. Thus, the NN predicts 1 if $P(y = 1) > 0.5$ and 0 otherwise. The activation functions in the hidden layers are decided arbitrarily, and they also affect the performance of the NNs. Three of the most common activation functions used in the hidden layers are: *logistic*, *tanh*, and *ReLu*. Table 3.2 defines each of these functions as well as the softmax function.

Name	**Function**
Logistic (sigmoid)	$\sigma(x) = \frac{1}{1+e^{-x}}$
Hyperbolic tangent (tanh)	$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Rectified linear unit (ReLu)	$relu(x) = max(0,x)$
Softmax	$softmax(x) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$

Table 3.2: A list of common activation functions used in neural networks.

NNs are typically trained with some variation of the *gradient descent* (GD) algorithm, which optimizes all weights W and b of a network N to minimize a given *loss function* $J(W,b)$. The loss function depends on the task being modeled by N. However, in supervised learning, one of the most common losses is the *cross-entropy*, which measures the difference between the training data distribution and the distribution modeled by N. Equation 3.1 formally defines the cross-entropy loss for a single example, where y_c is the target label for the class c, $\hat{y}_c$ is the output predicted by $N_{W,b}$ for class c, and C is the number of classes in the prediction task.

$$J(W,b) = -\sum_{c=1}^{C} y_c log(\hat{y}_c) \tag{3.1}$$

As shown in Algorithm 1, GD works by iteratively taking steps in the opposite direction of the gradient of the loss function $J(W,b)$ with respect to all weights. For a given number of iterations called *epochs*, GD (line 2) computes the gradient of $J(W,b)$ for the entire training dataset and (line 3) updates all weights W and b in the opposite direction of the gradient. The *learning rate* α is a parameter that controls the size of the training step. Computing the gradient (line 2) requires calculating the partial derivatives of the loss function with respect to all weights in N. This calculation is typically performed by an algorithm called *backpropagation*, which uses the chain rule to compute the gradient one layer at a time, iterating backwards from the output layer to avoid redundant calculations of intermediate terms in the chain rule.

Calculating the gradients for the whole dataset to perform just one update can be very slow or intractable for datasets that do not fit in memory. *Stochastic Gradient Descent* (SGD) is a variation of GD that solves this problem by splitting the training data into sets, called *batches*, and performing a training step for each batch. Although SGD supports training with

Algorithm 1 Gradient Descent

Require: Dataset (X, Y), a loss function $J(W, b)$, a NN $N_{W,b}$ with parameters W and b, the number of epochs e and the learning rate α.

Ensure: Updated parameters W and b that minimize the loss function $J(W, b)$.

1: **for** $i \leftarrow 1$ **to** e **do**

2: $\partial W \leftarrow \frac{\partial J}{\partial W}, \partial b \leftarrow \frac{\partial J}{\partial b}$

3: $W \leftarrow W - \alpha \partial W, b \leftarrow b - \alpha \partial b$

4: **end for**

very large datasets, it introduces convergence issues due to the variance in the frequent updates that cause the value of the loss function to fluctuate. *Adaptive Moment Estimation* (Adam) is a recent variation of SGD that mitigates this problem by having a learning rate per weight and separately adapting them during training [77]. In practice, most practitioners use the Adam optimizer, given that successful NNs typically require large datasets that do not fit in memory[2].

3.2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are an important architecture for AMC because they were specially designed to model sequential data. RNNs process sequences $\{x_1, x_2, \cdots, x_t\}$ step-by-step by keeping an internal state h_t that is updated every step. Each element x_i is a token (e.g., words in English or pitch classes in western music) traditionally encoded as a one-hot vector. Figure 3.3 shows an abstract diagram of a RNN. On the left-hand side, the RNN is shown with an input layer that passes a token x_t to a hidden layer A that updates h_t. A loop in

[2]The term *big data* is typically used to refer to these very large datasets.

the hidden layer allows information to be passed from one step of the network to the next. The right-hand side shows an unrolled version of the same RNN. The output layer of the network is omitted from the diagram because RNNs can produce one output per time step or one single output at the very last time step. This configuration depends on the learning task. However, the output layer typically maps the hidden state h_t to an output vector y_t.

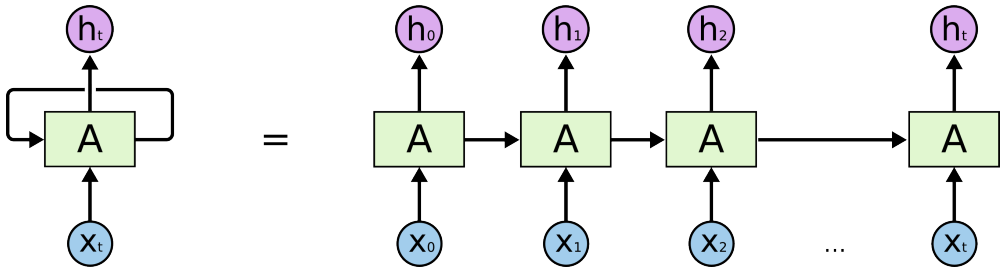


Figure 3.3: Diagram of a RNN [103].

Two of the most simple RNNs are the *Elman* network [36] and the *Jordan* network [71]. They are composed of an input layer, a single hidden layer, and an output layer. As shown in Equation 3.2, these RNNs produce an output y_t for each time step t. The matrices W_{xh} and W_{hh} represent the weights of the hidden layer, and the matrix W_{hy} represents the weights of the output layer. The functions f_h and f_y are the activation functions of the hidden and output layers, respectively. The only difference between these two networks is that, in the Elman network, the weights W_{hh} are fed from the hidden layer and, in the Jordan network, from the output layer.

Elman network	**Jordan network**	
$h_t = f_h(W_{xh}x_t + W_{hh}h_{t-1})$	$h_t = f_h(W_{xh}x_t + W_{hh}y_{t-1})$	(3.2)
$y_t = f_y(W_{hy}h_t)$	$y_t = f_y(W_{hy}h_t)$	

Todd [135] presented one of the first applications of RNNs for AMC: a *Jordan* network designed to generate melodies. The input of this network is a melody encoded as a sequence of pitch classes (e.g., *CDEGFEDF*), and the output is a single pitch. This network was trained to reconstruct given example melodies. Each melody contributed to t training steps, where t is the size (number of pitches) of the melody. At every training step t, the current example melody is given as input, and the network error is calculated by comparing the network output y_t with the respective pitch x_t of the input melody. After training, one can give new melodies as input to the network, which outputs new pitches by interpolating between the melodies seen during training. Duff [31] presented another early example of Jordan network for melody generation. However, instead of encoding melodies as a sequence of pitch classes, Duff [31] encoded them as a sequence of note intervals[3].

One of the major problems of RNNs consists of modeling long-term dependencies between symbols in a sequence. Modeling long-term dependency consists of creating a RNN capable of considering previous symbols that are distant from the one that is being predicted. In practice, simple RNNs are unable to connect the information between symbols that are very far from each other [8]. In music, modeling long-term dependencies is critical to generate long complete pieces with coherent form.

3.2.2 Long Short-Term Memory Networks

Long Short-Term Memory (LSTM) networks [58] are a special type of RNN explicitly designed to solve the long-term dependency problem. LSTMs also process sequences

[3]The distance in pitch between two notes.

$x = \{x_1, x_2, \cdots, x_t\}$ step-by-step by keeping an internal state h_t that is updated every step. However, as shown in Figure 3.4, the hidden layers A have a different structure allowing LSTMs to capture longer dependencies in the input sequences. A single LSTM module t is composed of an extra state C_t called *cell state*, which is responsible for carrying information through the entire LSTM network.

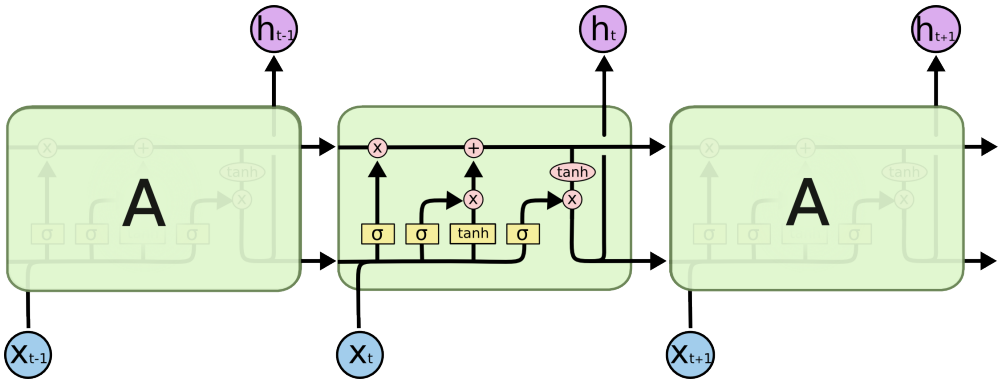


Figure 3.4: Diagram of an LSTM [103].

The flow of information in the cell state is controlled by three *gates*: an input gate i_t, an output gate o_t and a forget gate f_t. These gates facilitate the cell to remember or forget information for an arbitrary amount of time. Equation 3.3 formally defines the computation performed in each LSTM module, where W_f, W_i, W_c, and W_o are weight matrices, $[h_{t-1}, x_t]$ is the the hidden state vector h_{t-1} concatenated with the input vector x_t, and $\tilde{C}_t$ is the candidate vector to be added to the cell state. Each gate i_t, o_t, and f_t have the exact same equation, just with different weight matrices (W_i, W_o and W_f, respectively). The cell state C_t combines the input and forget gates to control the amount of information that will be included from the input versus the amount of information that will be forgotten from the current cell state, respectively.

The output gate controls the parts of the cell state that will be included in the final hidden state h_t. Modern RNNs (including LSTMs) typically have an extra layer, called *embedding* layer, that is added before any other hidden layer to transform the one-hot input vectors x_t into a dense vector representation called *embeddings*. In NLP, a word *embedding* is a learned representation for text where words that have the same meaning have a similar representation.

$$
\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, x_t]) \\
i_t &= \sigma(W_i[h_{t-1}, x_t]) \\
\tilde{C}_t &= tanh(W_c[h_{t-1}, x_t]) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma(W_o[h_{t-1}, x_t]) \\
h_t &= o_t * tanh(C_t)
\end{aligned}
\tag{3.3}
$$

Modern LSTM-based AMC systems typically train an LSTM as a LM, i.e. to predict the next token x_t given prefix tokens $x = \{x_1, x_2, \cdots, x_n\}$. Thus, the input token x_t at each time step t is mapped, with a *softmax* activation function, to a probability distribution $\hat{y}_t$ over the symbols defined in the music vocabulary. The LSTM is then trained with the cross-entropy loss function, which compares the predicted probability distribution $\hat{y}_t$ with the true next token x_t. For example, BachBot [86] uses an LSTM to generate polyphonic music in the style of Bach's chorales. BachBot was trained with the JSB Chorales dataset [14], where each chorale was encoded with a sequence of sixteenth-note frames. Each frame consists of four tuples (soprano, alto, tenor, and bass) in the form (*pitch*, *tie*), where *pitch* represents a MIDI pitch number, and *tie* is a boolean value that distinguishes whether a note is tied with another note at the same

pitch from the previous frame or is articulated at the current time step. DeepBach [52] is similar system that use LSTMs to geneate Bach chorales. The main difference between DeepBach and BachBot is that, in DeepBach, LSTMs consider both past and future contexts to predict the next token, while BachBot considers only the past.

Mao et al. [91] built upon a biaxial LSTM [69] to create a system called DeepJ, which can compose music conditioned on a specific mixture of composer styles. DeepJ uses a piano roll representation augmented with dynamics information, where the style of the music piece is encoded as a one-hot representation over all artists in the training data. Oore et al. [105] proposed another LSTM that can generate music with dynamics. They trained an LSTM on the piano pieces from the International e-Piano Competition [1] with a new encoding method that extracts tempo and velocity information from MIDI messages.

3.2.3 Transformers

Transformers [137] are modern architectures for sequence modeling based on *attention* mechanisms. In neural NLP models, an attention mechanism is a part of a NN that dynamically highlights relevant tokens of the input sequence [6]. Instead of keeping an internal hidden state that is updated at each time step like RNNs, transformers process entire sequences at once, associating an *attention* score to each input token, which determines how much that token contributes to the output. Because transformers process tokens in parallel, they can take advantage of the parallel computing offered by GPUs, and hence transformers can be trained considerably faster than LSTMs [137]. One drawback of transformers is that they can only process sentences with a fixed size instead of LSTMs that can process sentences of any size.

Transformers were originally designed in the context of *machine translation*, an NLP task that consists of translating a sequence from one language (e.g., English) to another (e.g., French). Machine translation is a *sequence-to-sequence* problem, where a sequence input x has to mapped into an output sequence y. NNs designed for machine translation normally have a *encoder-decoder* structure. The first part of the network, called the *encoder*, takes a sequence as input x and outputs a vector representation e (called *encodings*) of the input x. The second part, called *decoder*, takes the encodings e as input and outputs a sequence y.

As shown in Figure 3.5, the transformer has an encoder-decoder structure (the encoder is showed on the left side and the decoder on the right side). The transformer takes as input a sentence typically encoded with one-hot vectors and transforms it into two sequences: a sequence of *input embeddings* and a sequence of *positional encodings*. The former is a dense vector representation of words learned from the sparse one-hot input. The latter is a dense vector representation of the words' positions learned from the indices of the words in the sentence. The transformer adds the input embeddings and positional encodings together and passes the result through the encoder.

The encoder converts the (input + position) embeddings b into encodings e using a stack of n identical layers called *transformer blocks*. Each transformer block has two layers: a *multi-head attention layer* and a fully connected *feedforward layer*. A residual connection [54] is applied around each of the two layers, followed by a layer normalization [5]. A residual connection is a connection between non-contiguous layers. Layer normalization normalizes the activations of the previous layer, i.e. it applies a transformation that maintains the mean activation within each example close to 0 and the activation standard deviation close to 1.

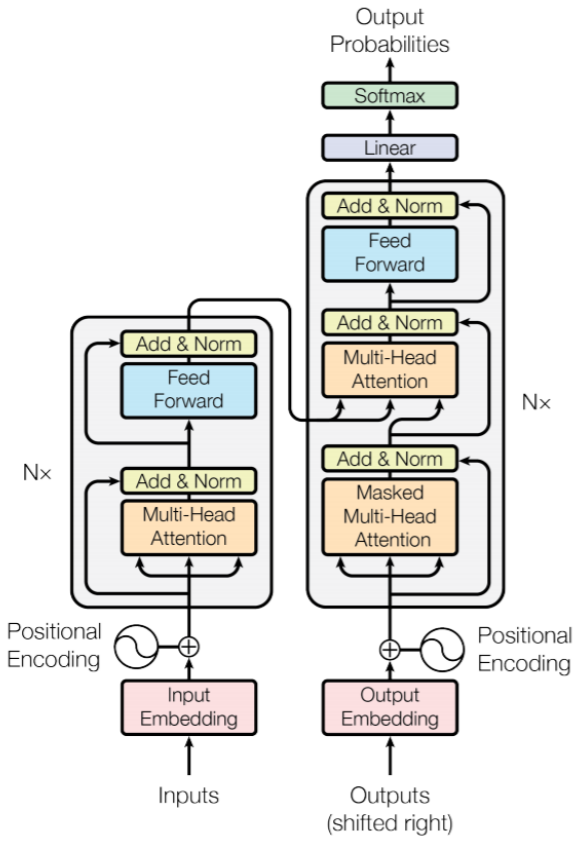


Figure 3.5: Diagram of a transformer [137].

The key component of the transformer is the *multi-head attention layer*, which computes a score matrix Z from the embeddings b or encodings e. The scores in Z represent the relationship between different words in the input sentence. For example, consider the sentence "The animal didn't cross the street because it was too tired.". In this sentence, the word "it" is related to "animal", and so when the transformer is processing the word "it", self-attention allows it to associate "animal" with "it". Figure 3.3 shows the encoder self-attention distribution for the word "it" in this example.

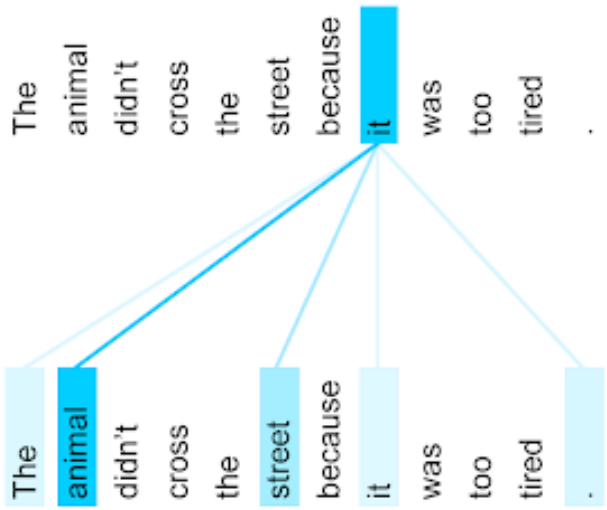


Table 3.3: The encoder self-attention distribution for the word "it" in the sentence "The animal didn't cross the street because it was too tired." [137].

The score matrix Z is computed similarly to a dictionary lookup: it takes a *query* matrix Q, a *key* matrix K, and a *value* matrix V, and outputs a weighted sum of the values that correspond to the keys that are most similar to the query. One of the most common self-attention mechanisms in a transformer is the scaled dot-product attention, which is shown in Equation 3.4. The matrices Q, K, and V are created by packing the embeddings b (or encodings e) of all the words in the input sentence into a matrix E, and multiplying it by the weight matrices W_q, W_k and W_v that are learned during training. The size d_k of the attention keys is a hyperparameter chosen according to the problem at hand.

$$
\begin{aligned}
Z = Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \\
Q = E * W_q \\
K = E * W_k \\
V = E * W_v
\end{aligned}
\tag{3.4}
$$

The transformer decoder is similar to the encoder, but it has an extra *masked multi-head attention* layer prepended to the transformer block to perform attention over the target output sentence y. This extra layer uses a mask to ensure that the predictions for position i depend only on the known outputs at positions less than i. The output of the decoder is computed with a linear layer followed by a softmax activation.

One can use the transformer decoder (without the encoder) to train a LM capable of generating sequences (e.g., text or music) similarly to an LSTM LM (see Section 3.2.2). Equation 3.5 formally defines a decoder-based transformer LM, where $x = \{x_1, x_2, \cdots, x_{t-1}\}$ is the input sequence, n is the number of transformer blocks (hidden layers), W_i is input embedding weight matrix, W_p is the positional encoding weight matrix, and $\hat{y}$ is the predicted probability distribution of the next token x_t given the input $\{x_1, x_2, \cdots, x_{t-1}\}$.

$$
\begin{aligned}
h_0 = W_i x + W_p \\
h_l = transformer\_block(h_{l-1}) \forall i \in [1,n] \\
\hat{y} = softmax(h_n W_e^T)
\end{aligned}
\tag{3.5}
$$

Transformers are currently the state-of-the-art of both natural and music language modeling. For example, Radford et al. [114, 115] proposed a series of models called GPT

(General Pre-trained Transformer), GPT-2, and GPT-3 that used the transformer decoder to create a model of natural language. Besides generating long coherent sequences of text, pre-trained GPT models can be fine-tuned to perform specific NLP tasks (e.g., commonsense reasoning, question answering, and summarization) with state-of-the-art performance [115]. Pre-training consists of training the GPT model as a (unsupervised) LM with a huge general dataset (e.g., Wikipedia). Fine-tuning is performed by stacking extra layers onto the pre-trained model and training these layers with a smaller (supervised) labeled dataset explicitly created for the task.

Music Transformer [63] is one of the first transformer-based LMs designed for AMC. It uses a new relative attention mechanism that improves memory consumption of the original decoder, allowing it to process longer sequences. Music Transformer achieved state-of-the-art performance on the MAESTRO dataset [53]. Donahue et al. [29] showed that a transformer can also compose multi-instrument scores by training it with the NES MDB [28] dataset. Donahue et al. [28] used a transfer learning procedure similar to Radford et al. [114], where they first pre-trained the transformer with the Lakh dataset (multiple instruments) and then fine-tuned it with the NES-MDB (4 instruments). They manually defined a mapping between the instruments from the two datasets. Pop Music Transformer [65] is a transformer model with a specialized music representation to compose pop piano music. It was shown to generate a better rhythmic structure than previous transformer models.

3.2.4 Variational Autoencoders

Variational autoencoders (VAEs) [78] are another modern architecture that can be used to generate music. VAEs are different from RNNs and Transformers because they were

not specifically designed to model sequences. Instead, they are generative models that can potentially learn to represent data in any domain. VAEs have an architecture similar to a traditional *autoencoder*, which is an encoder-decoder NN used to learn efficient encodings of unlabeled data (unsupervised learning). Figure 3.6 shows a diagram of an autoencoder.

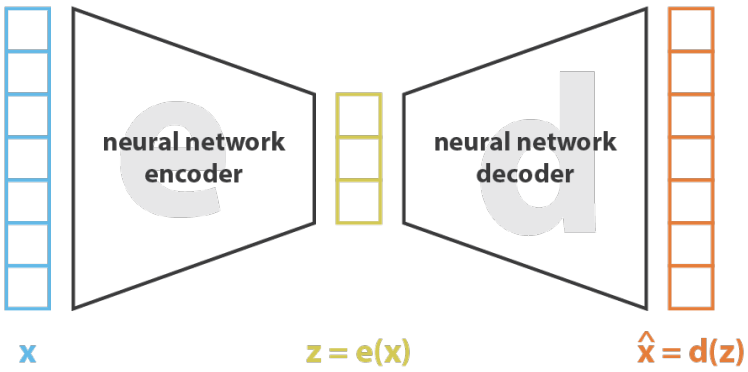


Figure 3.6: Diagram of an autoencoder [120].

An autoencoder builds a latent space of a dataset X with an encoder network e by learning to compress each example x into a vector z and then reproducing x from z with a decoder network d. A key component of an autoencoder is the bottleneck introduced by making the vector z have fewer dimensions than the input x, which forces the model to learn a compression scheme. During training, the autoencoder ideally distills the qualities that are common throughout the dataset. As shown in Figure 3.7, one can use an autoencoder as a generative model by sampling random vectors from the latent space learned by the encoder and using the trained decoder to build the output from the sampled vector.

One limitation of the autoencoder is that it often learns a latent space that is not con-

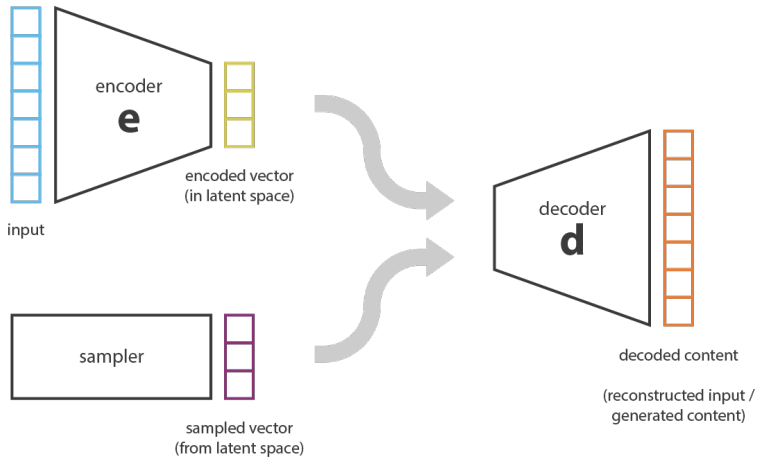


Figure 3.7: Sampling from the latent space built by an autoencoder [120].

tinuous[4] nor complete[5] [119]. This means that if one decodes a random vector sampled from the learned latent space, it might not result in a realistic output. VAEs solve this problem by encoding the dataset X as a probability distribution over the latent space instead of a single vector z. Typically, this distribution is assumed to be a multivariate normal distribution $N(\mu_x, \sigma_x)$. Figure 3.8 shows a diagram of a VAE. The difference between an autoencoder and a VAE is that the VAE encoder outputs two vectors μ_x and σ_x, instead of a single vector z. These two vectors represent the mean and standard deviation of a normal distribution N, respectively. The decoder d samples a vector $z \sim N(\mu_x, \sigma_x)$ from the distribution N and reconstructs the input x from z.

MusicVAE [118] is one of the first examples of VAE applied to AMC. It splits the input sequence x into U non-overlapping subsequences y_u, such that $x = \{y_1, y_2, \cdots, y_U\}$. The encoder processes the segmented input x with a bidirectional LSTM, whose hidden states are used to produce the latent distribution parameters μ_x and σ_x. The decoder is a novel hierarchi-

[4]Two close points in the latent space should yield similar outputs when decoded.
[5]Any point sampled from the latent space should yield a "meaningful" output when decoded

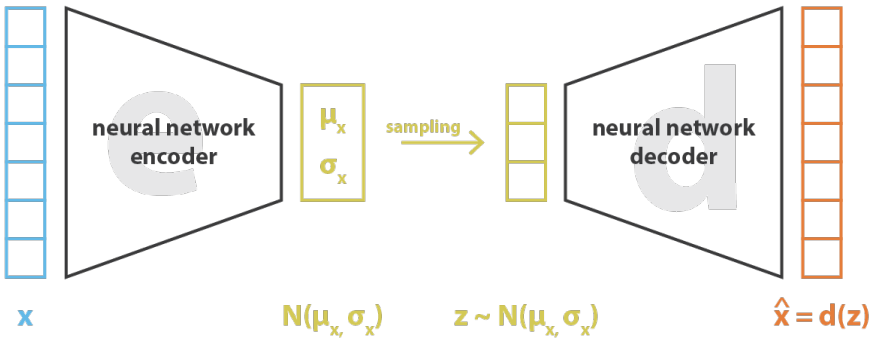


Figure 3.8: Diagram of a variational autoencoder [120].

cal LSTM that takes the latent vector $z \sim N(\mu_x, \sigma_x)$ as input and first produces U embedding vectors $e = \{e_1, e_2, \cdots, e_U\}$ (one for each subsequence y_u) with an LSTM called *conductor*. Each embedding vector is then passed through a fully connected layer to produce the initial states for a final decoder LSTM. The final LSTM then autoregressively produces a sequence of distributions over output tokens for each subsequence y_u via a softmax activation.

MIDI-VAE [19] uses a VAE to perform style transfer in polyphonic symbolic music. It works by separating a portion z_s of the latent vector z for style classification and another portion z_t to encode music structure (pitch, timber, and velocity). During generation, one can change the style s_i of a given piece x to another style s_j by passing x through the encoder to get z and swapping the values z_s^i and z_s^j. The modified latent vector is then passed through the decoder to get x with the target style s_j. Style labels can be music genres such as Jazz, Pop, and Classic; or composer names such as Bach or Mozart. VirtuosoNet [68] is a VAE designed to generate piano performances with expressive control of tempo, dynamics, and articulations. The encoder is a hierarchical LSTM that encodes music on different levels: note, beat, and

measure. The decoder renders musical expressions by first predicting the tempo and dynamics in measure level and then refining the result in note level.

3.2.5 Generative Adversarial Networks

Generative adversarial networks (GANs) [51] are another recent class of generative models that, in theory, can generate synthetic data in different domains. GANs are composed of two independent NNs: a *generator* and a *discriminator*. In its most basic form, the generator takes random noise as input and transforms it into a fake example. The discriminator is a binary classifier that discriminates examples as *fake* (0) or *real* (1). The generator and discriminator architectures depend on the generative task that one wants to perform with the GAN. For example, Figure 3.9 illustrates a GAN for handwritten digit generation. The random noise is a matrix M representing a grayscale image, which can be generated by sampling values between 0 (black) and 1 (white) from a uniform distribution. Typically, the generator uses convolutional layers to transform M into a fake handwritten digit. The discriminator then combines real and fake images to learn how to separate images between fake and real.

Training a GAN consists of training the generator and the discriminator together iteratively in alternating periods:

1. **The discriminator trains for one or more epochs.**

   The discriminator is trained with a loss function that penalizes it for misclassifying a real instance as fake or a fake instance as real. The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

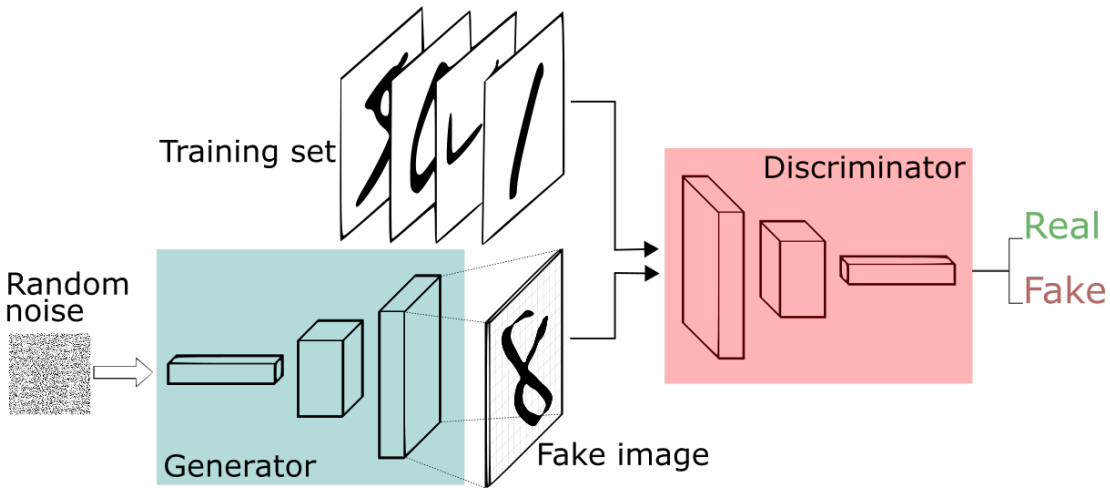


Figure 3.9: Diagram of a generative adversarial network [124].

2. **The generator trains for one or more epochs.**

   The generator is trained with a loss function that penalizes it for producing a sample that the discriminator classifies as fake. In other words, the generator loss is computed via the discriminator. Thus, the generator updates its weights through backpropagation from the discriminator to the generator.

The original GAN uses a single loss function called *minimax loss* to train both the generator and the discriminator [51]. Minimax loss is derived from the cross-entropy between the real and generated distributions. The generator tries to minimize this loss while the discriminator tries to maximize it. During training, G learns to generate fake data that resembles the original data, and D learns to distinguish the generator's fake data from real data. The training process aims to have a generator G that produces output that can fool the discriminator D. After training, one can generate new data points by sampling from different distribution points

learned by G.

GANs work well to generate continuous data such as images [17], but it has limitations in generating categorical data, especially in a sequential form such as text or music. Images can be represented by a continuous matrix M and so applying transformations (e.g., $M' = M + 0.05$) to M still results in defined images M' that can be classified as real or fake. In sequential domains such as text and music, tokens are encoded using embedding vectors. Applying transformation to an embedding vector does not necessarily generate a valid token. For example, assume that the word "university" is represented by the vector $v = [0.44, 0.37, -0.28]$. If one applies the transformation $v' = v + 0.05$ to the original v, the new vector v' not necessarily represent some word in the vocabulary. Therefore, updating the weights of G with the gradients of the minimax loss might lead G to generate invalid data. Another problem is that the discriminator can only provide feedback on entire sequences.

Different approaches have been proposed to solve these problems in the domain of symbolic music generation. For example, C-RNN-GAN [97] encodes MIDI files as a sequence of continuous note events. Each note is a tuple (l, p, i, dt), where l is the note length, p is the pitch frequency, i is note intensity, and dt is the time elapsed since the last note. With this encoding scheme, C-RNN-GAN used RNNs for both the generator and the discriminator. SeqGAN [150] combined adversarial training and reinforcement learning to generate monophonic music with a RNN generator and a convolutional discriminator. MidiNet [147] encodes MIDI files as a sequence of fixed-size piano rolls $M \in \{0,1\}^{128*w}$, where w is the number of time steps in each piano roll. Since each piano roll can be seen as a grayscale image, SeqGAN uses convolutional layers in the generator and the discriminator. MuseGAN [30] also uses a

piano roll encoding with a convolutional generator and discriminator to generate polyphonic music. Muhamed et al. [100] proposed a GAN where both the generator and the discriminator are transformers. The music pieces are encoded with the Music Transformer encoding scheme [63], and the Gumbel-Softmax trick [67] is used to address the gradient problem of categorical generators.

3.3 Decoding

As discussed in the previous section, most neural generative models for AMC are based on RNNs, LSTMs, or Transformers[6]. These sequential models use a softmax activation function in the output layer to create a LM $L = P(x_t|x_1,\cdots,x_{t-2},x_{t-1})$, where $\{x_1,\cdots,x_{t-2},x_{t-1}\}$ is an input sequence and x_t is the next token in that sequence. Typically, an *autoregressive* strategy is used to generate music with L, i.e. to *decode* the softmax output into a sequence of music tokens. One starts with a prior sequence of tokens $x = \{x_1,x_2,\cdots,x_{t1}\}$, which is fed into L to generate $L(x) = x_t$. Next, x_t is concatenated with x and the process repeats until a special end-of-piece token is found or a given number of tokens is generated. As defined in Equation 3.6, autoregressive generation assumes that the probability distribution of a sequence of tokens can be decomposed into the product of conditional next token distributions.

$$P(x_t|x_1,\cdots,x_{t-2},x_{t-1}) = \prod_{t=1}^{T} P(x_t|x_{1:t-1}) \tag{3.6}$$

Currently, most prominent autoregressive strategies for music (and text) decoding are based either on sampling or searching. While sampling is well suited for creative tasks such

[6]Including VAEs and GANs, which typically use RNNs, LSTMs, or Transformers as building blocks.

as music composition, searching better fits generative problems where specific solutions are expected, such as machine translation.

3.3.1 Top-k Sampling

In its most basic form, sampling consists of randomly picking the next token according to the conditional probability distribution given by the LM: $x_t \sim P(x_t|x_{1:t-1})$. Figure 3.12 shows and example of text generation with the prior sequence $x = \{The\}$. In the first step, the word *car* is sampled from the condition probability distribution $P(x_1|The)$ and, in the second step, the word *drives* is sampled from $P(x_2|The, car)$.

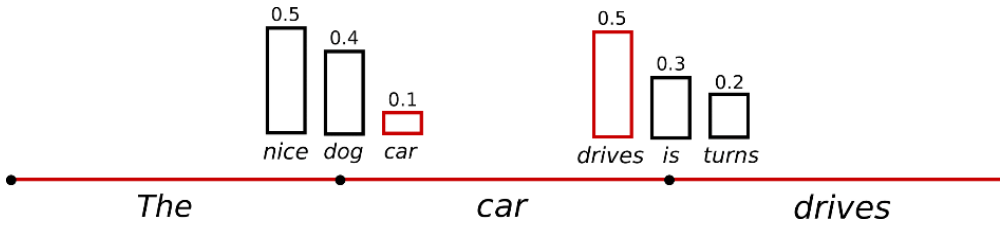


Figure 3.10: Example of sampling with prior sequence $x = \{The\}$ [138].

A simple trick called *temperature* can be applied to control the confidence of the LM. It consists of dividing the LM output before the softmax activation by a parameter $t > 0$. Lower temperatures $t < 1$ make the model increasingly confident in its top choices, while $t > 1$ decreases confidence. Figure 3.11 shows the previous example with temperature $t = 0.7$. The conditional probability $P(x_1|The)$ of the first step becomes more confident, leaving almost no chance for the word *car* to be selected. Thus, the word *nice* is sampled first, followed by the word *house*.

Top-k sampling [39] is another way to control the probability distribution of the LM.

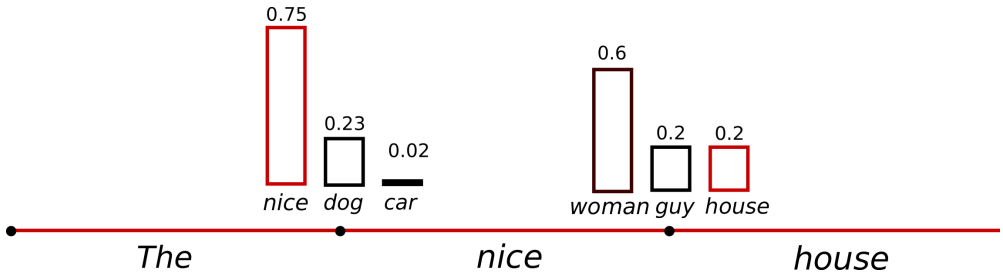


Figure 3.11: Example of sampling with prior sequence $x = \{The\}$ and temperature $t = 0.7$ [138].

It consists of using only the k most likely tokens in the distribution, redistributing the probability mass among only those *top-k* tokens. With this approach, the LM is filtered at each generation step and the token is picked randomly according to the resuling probability distribution. Figure 3.12 illustrates the previous example with top-k sampling ($k = 6$).

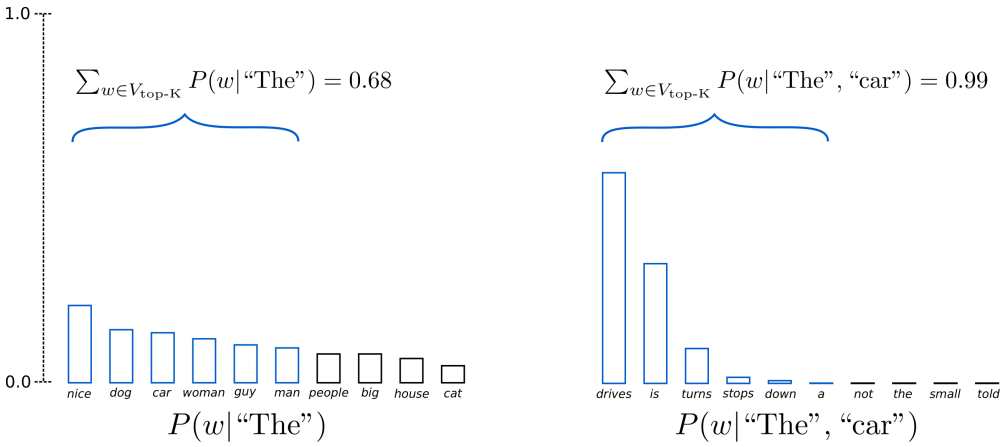


Figure 3.12: Example of top-k sampling with $k = 6$.

In step $t = 1$, the top-k sampling keeps the six words $\{nice, dog, car, woman, guy, man\}$ in the sampling pool, which encompass only two thirds of the probability mass. The words $\{people, big, house, cat\}$ are eliminated, even though they seem like reasonable candidates.

In step $t = 2$, the top six words represent almost all of the probability mass. Two of the selected words $\{down, a\}$ are arguably bad candidates. Nevertheless, the eliminated words $\{not, the, small, told\}$ are rather bad candidates and hence successfully eliminated. This example highlights that top-k sampling can jeopardize the LM, making it produce wrong sentences for sharp distributions and limiting the model's creativity for flat distributions.

3.3.2 Top-p (Nucleus) Sampling

Holtzman et. al. [60] proposed *top-p (nucleus) sampling* to address the degeneration problems faced by top-k sampling (and other decoding strategies). Instead of limiting the sample pool to a fixed size k, top-p samples from a dynamic *nucleus*, i.e. the smallest set of tokens whose cumulative probability exceeds a given probability p. Thus, the size of the sample pool is dynamically adjusted for each step depending on the LM distribution. Figure 3.13 shows the previous example with top-p sampling ($p = 0.92$).

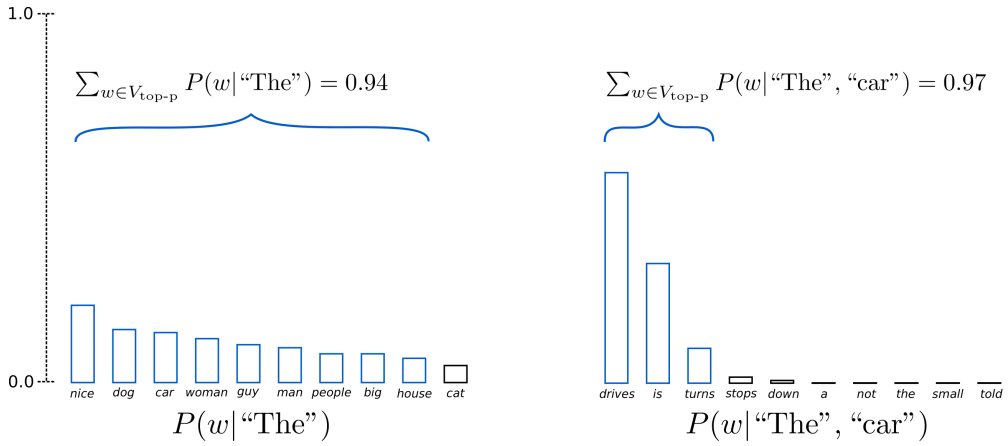


Figure 3.13: Example of top-p sampling with $p = 0.92$ [138].

The nucleus for $p = 92\%$ includes the nine most likely words in the first step and only three words in the second step. This example highlights that top-p sampling keeps a wide range of tokens in less predictable situations (e.g., step $t = 1$) and a few tokens in more predictable situations (e.g., step $t = 2$). Although top-p is theoretically more interesting than top-k, both methods work well in practice. Top-p can also be combined with top-k to avoid very low ranked tokens while allowing for some dynamic selection.

3.3.3 Greedy Search

Greedy search selects the token with the highest probability at each generation step t: $x_t = argmax P(x_t|x_{1:t-1})$. Figure 3.14 shows a text generation example with greedy decoding and prior sequence $x = \{The\}$. In step $t = 1$, greedy search selects the word *nice*, which has the highest probability among the three options $\{dog, nice, car\}$. In step $t = 2$, the options are $\{woman, house, guy\}$ and hence the greedy search selects the word *woman*. The final generated sentence is *The nice woman* with a joint probability of $0.5 * 0.4 = 0.2$.

The problem with greedy search is that it misses high probability tokens "hidden behind" low probability ones. In this example, the global optimal solution is the sentence *The dog has* (with joint probability 0.36), but the word *nice* has higher probability than *dog* in step $t = 1$. Thus, greedy search selects *nice* and completely disregards the branch with the word *dog* in step $t = 2$.

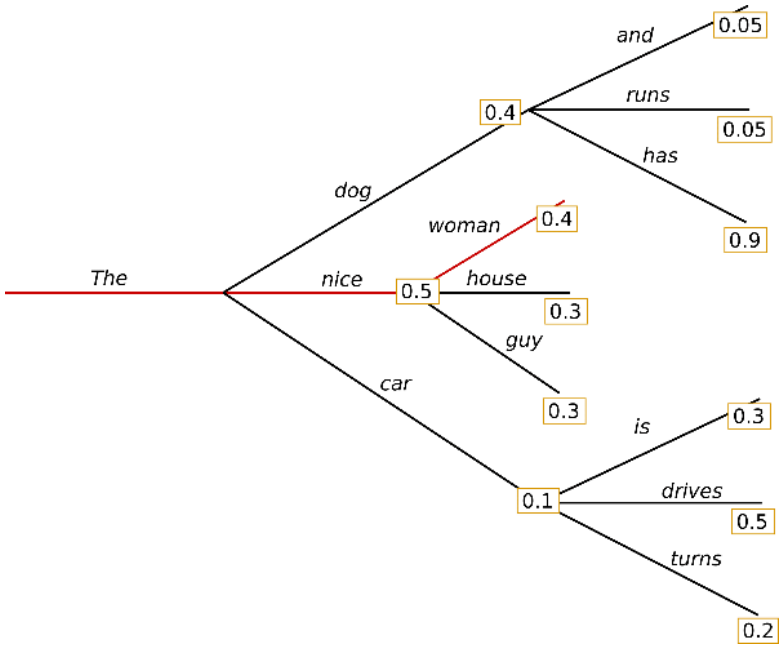


Figure 3.14: Example of greedy search with prior sequence $x = \{The\}$ [138].

3.3.4 Beam Search

Beam search reduces the risk of missing hidden high probability tokens by keeping the most likely b solutions (called *beams*) at each time step, where b is a parameter called *beam width*. At the final generation step, the solution (or beam) with the highest joint probability is selected. Figure 3.15 shows how beam search is capable of finding the best solution of the previous example with beam size $b = 2$.

In step $t = 1$, the two most likely sub-sequences are $\{The, dog\}$ and $\{The, nice\}$. In step $t = 2$, beam search expands the beam from the previous step with the two most likely sub-sequences $\{The, dog, has\}$ and $\{The, nice, woman\}$. At the end of the second step, beam search returns the beam with highest probability, which is *The dog has* with joint probability

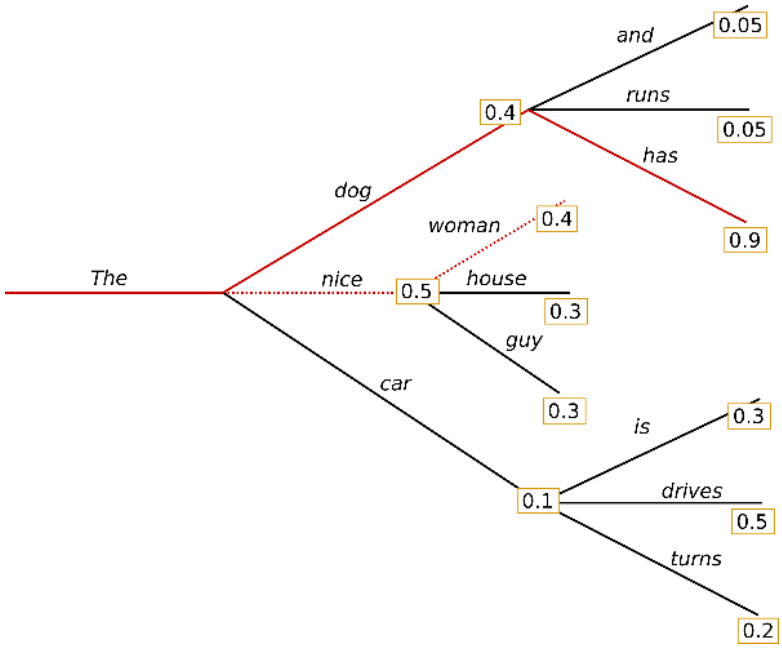


Figure 3.15: Example of beam search with prior sequence $x = \{The\}$ and beam size $b = 2$ [138].

0.36. The solutions generated with beam search are always as good as or better (more likely according to the LM) than the sequences generated with greedy search. However, beam search is not guaranteed to find the optimal solution.

A common problem with beam search (and other search-based approaches) is that the decoded sequences tend to become repetitive in a few generation steps [60]. This problem can be alleviated by combining beam search with sampling (including top-k and top-p) to create a *stochastic beam search* [112]. In this case, the sub-sequences are sampled at each time step according to their joint probability, instead of selected greedly.

Different *decoding* strategies (e.g., top-k sampling and beam search) can be used to generate music with different NN *architectures* (e.g., LSTM, Transformer, VAEs, and GANs) trained with different *symbolic music* datasets (e.g. MAESTRO, Lakh, and JSB Chorales). This

chapter presented an overview of these fundamental ideas of deep learning for music generation. This dissertation builds upon these ideas for controlling the emotion of music composed with neural LMs. The next chapter reviews previous works that are related to this dissertation, including methods to control neural LMs and other AMC systems that generate music with a target emotion.

Chapter 4

Controllable Algorithmic Music Composition

Deep generative models are currently the leading method for AMC [148]. However, a major problem with this method is controlling the generation process towards a given compositional goal. For example, controlling a model trained on classical piano pieces to compose a tense piece for a horror movie scene. Controlling emotions in AMC has been actively investigated in the field of *affective algorithmic composition* [141]. Therefore, this chapter presents a literature review of this field, including models to represent emotion and methods to control emotion in symbolic AMC systems. Given the similarity between text and music generation tasks, this chapter also covers NLP methods to control neural natural language models.

4.1 Affective Algorithmic Composition

Affective algorithmic composition (AAC) is a research field concerned about controlling AMC methods to compose music that makes listeners perceive or feel a given target emotion [141]. AAC is essential to a variety of applications, ranging from soundtrack generation

[140] to sonification [21] and music therapy [94]. The AAC community has explored various methods to compose music with a target emotion algorithmically: expert systems [140], evolutionary algorithms [75], Markov chains [98], deep learning [90], among others. These methods have used different *models of emotion*, but most of them are concerned with controlling *perceived emotions* instead of *felt emotions*. Therefore, they are normally evaluated with a listening test (i.e., a user study) or a qualitative analysis of generated samples. This section introduces the most common models of emotion used in the AAC literature and the different approaches to control emotion in AMC. It also highlights the different methodologies used in the AAC literature to conduct the qualitative analysis and the listening tests.

4.1.1 Models of Emotion

The study of affective phenomena has a very dense literature with multiple alternative theories [34]. This section does not aim to discuss all of them but to provide a definition of emotion that is useful for designing neural LMs for AAC. According to Williams et al. [141], the literature concerning the affective response to musical stimuli defines emotion as a short-lived episode, usually evoked by an identifiable stimulus event that can further influence or direct perception and action. Williams et al. [141] also differentiate emotion from affect and mood, which are longer experiences commonly caused by emotions.

There are two main types of models used to represent emotions: *categorical* and *dimensional* [33]. Categorical models use discrete labels to classify emotions. For example, Ekman's model [35] divides human emotions into six basic categories: anger, disgust, fear, happiness, sadness, and surprise. This model builds on the assumption that an independent neu-

ral system subserves every discrete basic emotion [33]. Therefore, any other secondary emotion (e.g. rage, frustration, and grief) can be derived from the basic ones. Parrott [109] proposed a similar model, but deriving a hierarchical structure with the following six basic emotions: love, joy, surprise, anger, sadness, and fear. These basic emotions are expanded into secondary emotions (e.g. passion, pleasure, and envy), which in turn derive ternary emotions (e.g. compassion, frustration, and guilt). Some emotions are more present in the music domain and are evoked more easily than others. For example, it is more common for a person to feel happiness instead of disgust while listening to music. The Geneva Emotion Music Scale (GEMS) [151] is a categorical model specifically created to capture the emotions that are evoked by music. GEMS divides the space of musical emotions into nine categories: wonder, transcendence, tenderness, nostalgia, peacefulness, energy, joyful activation, tension, and sadness. These nine emotions group a total of 45 specific labels.

Dimensional models represent emotions as a set of coordinates in a low-dimensional space. For example, Russell [121] proposed a general-purpose dimensional model called *circumplex*, which describes emotions using two orthogonal dimensions: valence and arousal. Instead of an independent neural system for every basic emotion, the circumplex model assumes that all emotions arise from two independent neurophysiological systems dedicated to the processing of valence (positive–negative) and arousal (mild–intense). Figure 4.1 shows a graphical representation of the circumplex model. The *vector model* [15] is another two-dimensional model based on valence and arousal. It represents emotions as vectors, where valence is a binary dimension (positive or negative) that defines the vector's direction and arousal is a continuous dimension that defines the vector's magnitude.

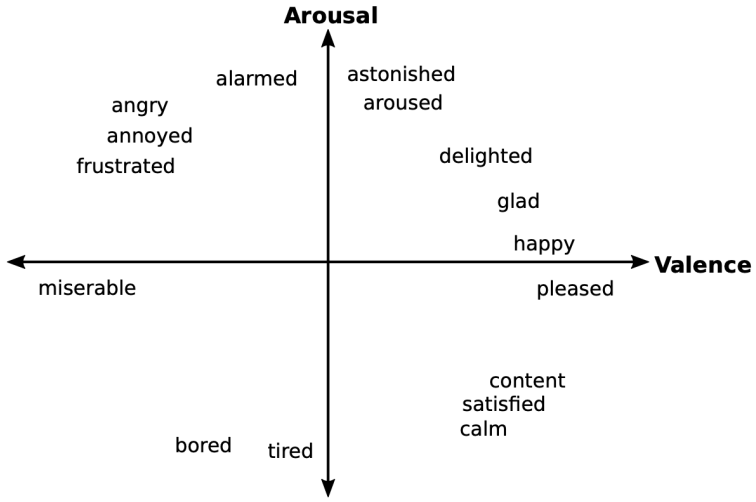


Figure 4.1: The circumplex model of emotion. The horizontal and vertical axes represent valence and arousal, respectively [121].

Both categorical and dimensional models have been widely used in AAC systems [141]. The choice of model is especially important for data-driven methods (e.g., neural networks) since they rely on datasets of music annotated by listeners according to the selected model. Given that listeners typically use words to describe the emotions they perceive in music, categorical models might be easier than dimensional models for this annotation task. On the other hand, dimensional models allow a more precise description of the perceived emotion. Such precision is particularly helpful to describe emotions in ambiguous pieces, where listeners can select intermediate values between the different basic emotions (e.g., happy and calm) perceived in the piece. Moreover, dimensional models can be easily mapped to categorical models. The VGMIDI dataset, created as part of this dissertation, used the circumplex model (dimensional) to exploit this flexibility of the dimensional methods, making the dataset more general and thus reusable for other future AAC systems.

Independent of the model, emotions can also be classified as *perceived* or *felt* [44]. A person *perceives* an emotion when they objectively recognize that emotion from their surroundings. For example, one can usually recognize someone else's emotion using expressed cues, including facial expression, tone of voice, and gestures. The same can happen when one listens to music, in that they recognize the music as happy or sad using cues such as key, tempo, or volume. A person *feels* an emotion when they actually experience that emotion themselves. For example, one typically experiences fear in response to someone else's anger. This example shows that a given perceived emotion can trigger a different felt emotion.

This dissertation focuses on generating symbolic music that listeners perceive to have a target emotion. Ideally, the listeners should also experience this emotion in response to their perception. However, emotions are less frequently felt in response to music than perceived as expressive properties of the music [151]. Moreover, symbolic music (e.g., MIDI) has limited expressivity compared to traditional recorded human performances (audio representation), making it harder for symbolic music to make listeners experience the target emotions. Focusing on perceived emotions facilitates evaluating the different methods proposed in this dissertation and still allows one to apply these methods to a wide range of problems.

4.1.2 Expert systems

Expert systems are one of the most common methods of AAC [141]. They encode knowledge from music composers to map musical features into a given (categorical or dimensional) emotion. For example, Williams et al. [140] proposed a system to generate sound-

tracks for video games from a scene graph [1] annotated according to twelve categorical emotions derived from the circumplex model. First, a second-order Markov chain learns to generate melodies from a symbolic music dataset. Then, an expert system transforms the melodies generated via the Markov chain to match the annotated emotions in the graph. The transformations are performed by mapping each of the twelve emotions to a different configuration of five music parameters: rhythmic density, tempo, modality (major or minor), articulation, mean pitch range, and mean spectral range. For example, a melody generated for a happy scene would be transformed to have high density, medium tempo, major mode, staccato articulation, medium mean pitch range, and high spectral range (clear timbre). Williams et al. [140] evaluated their system by qualitatively examining a few examples of generated pieces.

TransProse [24] is an expert system that composes piano melodies for novels. It splits a given novel into sections and uses a lexicon-based approach to assign an emotion label to each section. TransProse composes a melody for each section by controlling the scale, tempo, octave, and notes of the melodies with pre-defined rules based on music theory. For example, the scale of a melody is determined by the sentiment of the section: positive sections are assigned a major scale, and negative sections are assigned a minor scale. TransProse was evaluated with a qualitative analysis of nine music pieces generated by the system for nine respective novels.

Scirea et al. [123] presented a framework called MetaCompose designed to create background music for games in real-time. MetaCompose generates music by (i) randomly creating a chord sequence from a pre-defined chord progression graph, (ii) evolving a melody for this chord sequence with a genetic algorithm and (iii) producing an accompaniment, adding

[1] A graph defining all the possible branching of scenes in the game.

rhythm and arpeggio, for the melody/harmony combination. Finally, MetaCompose uses an expert system called *Real-Time Affective Music Composer* to transform the final composition to match a given emotion in the circumplex model. This expert system controls four musical attributes: volume, timbre, rhythm, dissonance. For example, low arousal pieces were controlled to have lower volume. Scirea et al. [123] evaluated each component of the MetaCompose with a pairwise listening test. The components of the system were systematically (one-by-one) switched off and replaced with random generation, generating different "broken" versions of the framework. These broken versions were paired with the complete framework and evaluated by human subjects according to four criteria: pleasantness, randomness, harmoniousness, and interestingness. For each criteria, the participants were asked to prefer one of two pieces, also having the options of *neither* and *both equally*.

Expert systems are a great approach for evaluating mappings between a small set of music features and emotions. However, given the multidimensional (melody, harmony, rhythm, orchestration, dynamics, etc.) nature of music, it is hard to create a large set of rules that consider all the dimensions. The challenge is not only in the number of rules one needs to make, but these rules might also contradict each other when combined together. The neural language models explored in this dissertation don't have these problems because neural networks learn can learn these rules from music data.

4.1.3 Evolutionary Algorithms

To control emotions in music generated with EAs, one has to define a fitness function that guides individuals encoding symbolic music towards a given emotion. It is very challeng-

ing to design a function that formally evaluates subjective aspects of music emotion. Thus, most EAs for AAC use interactive evaluation functions, where human subjects judge whether or not the generated pieces match a target emotion. For example, Kim and André [75] proposed an IGA to compose polyrhythms[2] for four percussion instruments. It starts with a random population of polyrhythms and evolves them towards relaxing or disquieting emotions. A polyrhythm is encoded with four 16-bit strings, one for each instrument. A single bit in the string represents a beat division where 1 means that a (unpitched) note is played in that division and 0 means silence. The fitness of a polyrhythm is given by a human subject who judges it as relaxing, neutral, or disquieting. The selection strategy keeps the four most relaxing and four most disquieting individuals for reproduction with one-point crossover and mutation. Results showed that the genetic algorithm generated relaxing polyrhythms after 20 generations while it took only 10 for it to generate disquieting ones.

Zhu et al. [153] presented an IGA based on the KTH rule system [43] to create affective performances of pre-composed music pieces. The KTH rules model performance principles within the realm of Western classical, jazz and popular music. These rules control different music performance parameters (e.g. phrasing, articulation, and tonal tension) with weights called *k values* that represent the magnitude of each rule. Zhu et al. [153] encoded the individuals as a set of k values used to create MIDI performances of pre-composed pieces according to the KTH rules. The genetic algorithm evolves a population to find optimal k values that yield performances that are either happy or sad. The fitness of the performances are given by human subjects with a seven-point Likert scale.

[2] A polyrhythm is the concurrent playing of two or more different rhythms.

Nomura and Fukumoto [102] designed a distributed IGA to generate four-bar piano melodies with controllable "brightness". Multiple human evaluators evolve independent populations of melodies in parallel. In some generations, the genetic algorithm exchanges individuals between the independent populations. With the exchange, evaluators are affected by each other and the solutions are expected to agree with everyone's evaluations. Each individual in a population represents a melody with a sequence of sixteen pitch numbers (as defined by the MIDI protocol). Each element in the sequence is mapped into a quarter note with the pitch defined by the element. Evaluators give the fitness of an individual based on a seven-point Likert scale, where 1 means "extremely dark", 4 means "neither", and 7 means "extremely bright". An experiment with ten parallel evaluators showed that after seventeen generations, the independent populations converged to similar melodies.

The benefit of interactive EAs is that when the population converges towards a target emotion, no further evaluation is needed to check if the generated pieces indeed match that emotion. These approaches are well suited for applications that require the human to be in the loop of the generative process (e.g., assisted composition tools). On the other hand, interactive EAs are model-free approaches, i.e., they only provide a set of solutions at the end of the evolutionary process. Every time one wants to generate a new set of pieces, the slow interactive evolutionary process must be restarted. The NNs explored in this dissertation don't have this problem because they are trained as language models which can generate as many music pieces as needed. Neural networks are well suited for real-time applications where the generated music is constantly changing (e.g., generative soundtracks).

4.1.4 Markov Chains

AAC systems based on Markov chains are typically hybrid systems where Markov chains compose an underlying piece of music that is transformed by an expert system. The work of Williams et al. [140] described earlier in this chapter is an example of such a system. Ramanto and Maulidevi [117] proposed a similar system in which the Markov chain is designed manually instead of derived from a corpus. There is very little research on training Markov chains directly from datasets of symbolic music labeled according to a model of emotion.

One of the few examples is the work of Monteith et al. [98, 20], which uses different Markov models to generate polyphonic music from a MIDI corpus labeled according to the Parrott model of emotion [109]. The corpus is composed by 45 movie soundtracks labeled by 6 researchers using the 6 basic Parrott emotions: love, joy, surprise, anger, sadness, and fear. The system is divided into four main components: a rhythm generator, a pitch generator, a chord generator, and an instrumentation planner. First, the rhythm generator randomly selects and transforms a rhythmic pattern from the subset of pieces with the target emotion. Second, the pitch generator assigns pitch values to the notes in the rhythm by sampling from a Markov chain trained with the pieces from the target emotion. Third, the chord generator generates an underlying harmony for the provided melody with a hidden Markov model. Finally, the instrumentation planner probabilistically selects the instruments for melodic and harmonic accompaniment based on the frequency of various melody and harmony instruments in the corpus.

The system uses single-layer feedforward networks, trained with the same MIDI corpus, to classify the outputs of the rhythm and melody generators as having the target emotion or

not. The system only accepts rhythms and melodies classified by the neural networks as having the target emotion. Monteith et al. [98] evaluated their system with a listening test in which 13 human subjects selected the emotion that they perceived in the pieces. Moreover, the subjects used two 10-point Likert scales to evaluate how human-like and unique the pieces are.

Markov chains are similar to the neural language models used in this dissertation as they both learn mappings from music features to emotions. However, as discussed in Chapter 2, low order Markov chains typically generate unmusical compositions that wander aimlessly, while high order ones tend to repeat segments from the corpus and are very expensive to train. On the other hand, the neural language models explored in this dissertation are currently the leading methods for algorithmic music composition [148].

4.1.5 Deep Learning

To train deep neural networks that can generate music with a target emotion, one needs a relatively large dataset of symbolic music labeled according to a model of emotion. Such datasets started to be created only recently, and this dissertation is part of the first wave of research in this area. The remainder of this section presents prominent deep learning approaches proposed to date. All of them use different datasets and have been developed concurrently with this dissertation. For example, SentiMozart [90] is a framework that generates piano pieces that match the emotion of a given facial expression. It uses a convolutional NN to classify the emotion of a facial expression and an LSTM to generate a music piece corresponding to the identified emotion. The convolutional NN was trained with the FER-2013 [50] dataset, which has 35,887 images of facial expressions labeled with the following emotions: angry, disgust,

fear, happy, sad, surprise, and neutral. To train the LSTM, Madhok et al. [90] created a dataset with 200 MIDI piano pieces, each labeled by 15 annotators according to the classes happy, sad, and neutral. Three LSTM were trained independently, one for each emotion. To unify the models of emotion between the two datasets, Madhok et al. [90] merged the categories sad, fear, angry, and disgust from the FER-2013 dataset into the sad emotion. The categories happy and surprise were mapped into the happy emotion. Thus, the output $e \in \{happy, sad, neutral\}$ of the convolutional NN selects the respective LSTM that, in turn, composes a piece conveying the emotion e. SentiMozart was evaluated with a listening test where 30 human subjects judged 30 randomly chosen images (10 of each class) and their corresponding generated pieces. The participants used a 11-point Likert scale in which 0 means sad, 5 means neutral, and 10 means happy.

Tan and Antony [133] proposed a deep NN for a similar problem: generating music that matches an emotion expressed by visual artworks (e.g. paintings, illustrations, and collages). They paired images of paintings with MIDI piano pieces labeled with the same emotion and used an encoder-decoder network as a generative model. The encoder is a pre-trained convolutional NN called ResNet [54]. Tan and Antony [133] compared two decoders: an LSTM and a (decoder) transformer. The music pieces were encoded as sequences of tokens with the MIDI-based method proposed by Oore et al. [105]. Tan and Antony [133] trained their model by pairing emotion-labeled images from You et al. [149] (17,349 images) with emotion-labeled music pieces from Panda et al. [107] (196 MIDI files). Since these two datasets use different categorical models of emotion, Tan and Antony [133] mapped the emotion labels of the music dataset to the labels of the image dataset. The trained model was evaluated with a listening

test and a machine classification test. In the listening test, six human subjects were asked to evaluate the sentiment of the music pieces and the images with a 10-point Likert scale without knowing which music piece was related to which image. The machine evaluation test consisted of training an emotional correspondence classifier to predict if a given pair of images and pieces express the same sentiment.

Zhao et al. [152] presented a conditional Biaxial LSTM [69] to generate piano pieces with controllable emotion. They labeled the *Piano midi.de* dataset according to a categorical model of emotion: happy, sad, peaceful, and tense. They used a piano roll representation for the input music pieces, which are processed by the LSTM together with an emotion signal associated with each time step of the input. This signal is encoded by an extra embedding layer and then added to the embedding of the input. Zhao et al. [152] evaluated the quality of the generated pieces according to four metrics of music structure: polyphony, scale consistency, 3-tone repetitions, and tone span. Moreover, they performed a listening test with 30 human subjects to evaluate if the subjects agree with the emotions intended by the model.

Music FaderNets [132] is a framework based on a *gaussian mixture variational autoencoder* (GM-VAE) that can learn high-level music feature representations (such as emotion) by modeling corresponding low-level structural music features. Based on Yang et al. [148], Music FaderNets learns the low-level features of rhythm z_r, note density z_d, and key z_k with separated RNN encoders e_r, e_d, and e_k, respectively. The high-level features are then inferred from the low-level representations (z_r, z_d, and z_k) via semisupervised clustering. Music Fadernets was trained to reconstruct pieces encoded as a sequence of tokens [105] extracted from

the Yamaha Piano-e-Competition dataset [1] and the VGMIDI dataset[3] [42]. Results showed that Music Fadernets successfully learns the intrinsic relationship between arousal (high-level feature) and its corresponding low-level attributes of rhythm and note density.

The main benefit of the neural LMs investigated in this dissertation over other AAC methods is that such LMs can learn the subjective mappings between music features and emotions from data. This makes them flexible and, in turn, applicable to different AAC problems. One just needs to create a dataset for the problem at hand. Neural LMs can also be used to generate a wide range of music pieces with the same trained model. Moreover, deep generative models are currently the leading method for AMC [148], which ensures that neural AAC systems can produce high-quality (i.e., similar to human composed pieces) music.

4.2 Controllable Neural Language Models

This dissertation is also related to methods that control natural LMs to generate text with given characteristics (e.g. topic, style, and sentiment). For example, Radford et al. [113] showed that fine-tuning a neural LM with an extra classification head to perform sentiment analysis exposes the neurons in the LM that carry sentiment signal. Particularly, when pre-training a character-level LSTM LM on Amazon product reviews [55] and fine-tuning it with L1 regularization on the *Stanford sentiment treebank* [128], most of the sentiment signal in the classification head comes from a single neuron in the LSTM. Therefore, this neuron can be manually adjusted to control the LM to generate new product reviews with a target sentiment.

CTRL [74] is a transformer LM trained to generate text conditioned on special tokens,

[3]The VGMIDI dataset is a contribution of this dissertation described in Chaper 5.

called *control codes*, that inform the LM about the characteristics (e.g. style) of the text to be generated. These control codes are derived automatically from the text source (e.g. Wikipedia), which means no manual annotation is needed. During training, every example sentence x is processed together with a set of control codes. At generation time, CTRL can produce text with a particular style s, for example, by conditioning the prior input x with a signal representing s.

Holtzman et al. [59] proposed a decoding strategy that uses a set of discriminators (NNs) to steer the probability distribution of a LM at generation time. At each decoding step, the probabilities of the next words given by the LM are multiplied by weighted scores of four different discriminators: repetition, entailment, relevance, and lexical diversity. The discriminators are trained independently with a dataset of pairs (x, y), where x is a prior context sentence, and y is the completion of that sentence. For each discriminator, the loss function measures the difference between the scores assigned to the truth continuation y and the scores assigned to the continuation generated by the LM. Once all the discriminators are trained, Holtzman et al. [59] optimize the weights used to combine the score of each discriminator.

The Plug and Play LM [23] combines a pre-trained LM with an attribute (e.g. sentiment) classifier C to guide text generation by fine-tuning the LM hidden layers at decoding time. At each generative time step, the LM hidden layers are shifted in the direction of the sum of two gradients: one towards the higher log-likelihood of the target attribute (as given by C) and the other towards higher log-likelihood of the unmodified LM. The shifting process occurs in three steps: (1) a forward pass using C to compute the likelihood of the target attribute, (2) a backward pass to update the LM hidden states with gradients from the attribute classifier C, and (3) another forward pass to update the distribution over the vocabulary from the updated LM

hidden layers. With this approach, multiple attribute classifiers can be combined at generation time with customized weights.

Given a music dataset labeled according to a model of emotion, one can apply the methods discussed in this section to control music LMs to generate pieces with a target emotion. The next chapter presents a method based on Radford et al. [113] to control the sentiment of piano pieces generated by an LSTM. The chapter after that discusses a variation of beam-search inspired by Holtzman et al. [59] to control a transformer to generate piano pieces with a target emotion.

Chapter 5

Learning to Generate Music with Sentiment

5.1 Introduction

As discussed in the previous chapter, Radford et al. [113] showed that fine-tuning a pre-trained LSTM LM with an extra classification head to perform sentiment analysis exposes the neurons in the LM that carry sentiment signal. For particular datasets, most of the sentiment signal in the classification head comes from a single neuron in the LSTM. Thus, this neuron can be manually adjusted to control the LSTM LM to generate text with a target sentiment. This chapter introduces a method inspired by Radford et al. [113] to generate symbolic music with a target sentiment (positive or negative). First, the VGMIDI dataset is introduced, a new dataset of symbolic piano pieces labeled according to the circumplex model of emotion. Second, an LSTM is trained as a LM with the unlabeled pieces of the VGMIDI dataset. Finally, this LSTM LM is fine-tuned with an extra linear layer and L1 regularization on the labeled pieces of the VGMIDI dataset. Different than the findings of Radford et al. [113], this fine-tuning step did

not expose a single sentiment neuron but many neurons that contribute to sentiment in a more balanced way. Thus, a genetic algorithm is applied to optimize the weights of these sentiment neurons, controlling the LSTM LM to generate either positive or negative music. This approach is evaluated with two experiments. First, a cross-validation on the VGMIDI dataset shows that the model obtains good prediction accuracy. Second, a listening test shows that human subjects agree that the generated music has the intended sentiment; however, negative pieces can be ambiguous. This work was published in the Proceedings of the 20th Conference of the International Society for Music Information Retrieval (ISMIR19) [42].

5.2 The VGMIDI dataset

A new dataset called VGMIDI has been created to apply the *sentiment neuron* [113] method to the symbolic music domain. This dataset contains piano arrangements of video game soundtracks in MIDI format. The VGMIDI dataset has had three major versions during this dissertation. The work reported in this chapter used the initial version, which had 823 pieces, varying in length from 26 seconds to 3 minutes. Among these pieces, 95 are annotated according to the circumplex (valence-arousal) model of emotion. VGMIDI uses the circumplex model because it allows continuous annotation of music, and because of its flexibility–one can directly map a valence-arousal (v-a) pair to a multiclass (e.g., happy, sad, tense, and peaceful) or a binary (positive/negative) model. Thus, the same set of labeled data permits the investigation of AAC as both a classification (multiclass/binary) or a regression problem. The circumplex model is also one of the most common models used to label emotion in music [129]. A few

similar datasets have been created concurrently with the VGMIDI [90, 133, 152], but none are labeled according to sentiment.

Annotating a piece according to the circumplex model consists of continuously listening to the piece and deciding what v-a pair best represents the emotion of that piece in each moment, producing a time series of v-a pairs. This task is subjective, and hence there is no single "correct" time series for a given piece. Thus, VGMIDI was labeled by asking several human subjects to listen to the pieces and then considering the average time series as the ground truth. This process was conducted online via Amazon Mechanical Turk, where each piece was annotated by 30 subjects using a web-based tool designed specifically for this task. Each subject annotated 2 pieces out of 95, and got rewarded USD $0.50 for performing this task.

5.2.1 Annotation Tool and Data Collection

The tool designed to annotate the VGMIDI dataset is composed of five steps, each one being a single web page. These steps are based on the methodology proposed by Soleymani et al. [129] for annotating music pieces represented as audio waveforms. First, participants are introduced to the annotation task with a short description explaining the goal of the task and how long it should take on average. Second, they are presented with the definitions of valence and arousal. On the same page, they are asked to play two short pieces and indicate whether arousal and valence are increasing or decreasing. Moreover, annotators are asked to write two to three sentences describing these short pieces. This page is intended to measure their understanding of the circumplex model and willingness to perform the task. Third, a video tutorial was made available to the annotators explaining how to use the annotation tool. Fourth, annotators are

exposed to the main annotation page.

The main page has two phases: calibration and annotation. In the calibration phase, annotators listen to the first 15 seconds of the piece to get used to it and define the starting point of the annotation circle. In the annotation phase, they listen to the piece from beginning to end and label it using the annotation circle, which starts at the point defined during the calibration phase. Figure 5.1 shows the annotation interface for valence and arousal, where annotators click and hold the circle (with the play icon) inside the circumplex model (outer circle), indicating the current emotion of the piece. In order to maximize annotators' engagement in the task, the piece is only played while they maintain a click on the play circle. In addition, basic instructions on how to use the tool are shown to the participants along with the definitions of valence and arousal. A progression bar is also shown to the annotators to know how far they are from completing each phase. This last step (calibration and annotation) is repeated for a second piece. All pieces the annotators listened to are MIDI files synthesized with the *Yamaha C5 Grand Piano* soundfont[1]. Finally, participants provide demographic information, including gender, age, location (country), musicianship experience, and whether they previously knew the pieces they annotated.

5.2.2 Data Analysis

The annotation task was performed by 1425 annotators, where 55% are female and 42% are male. The other 3% considered themselves as transgender female, transgender male, genderqueer, or choose not to disclose their gender. All annotators are from the United States

[1] `http://freepats.zenvoid.org/Piano/acoustic-grand-piano.html`

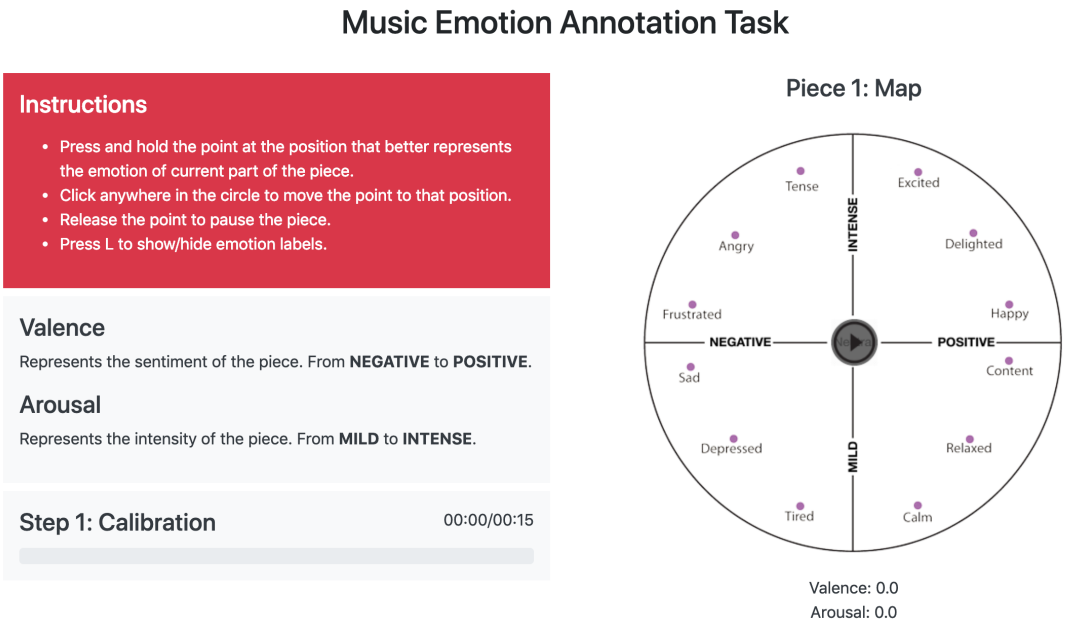


Figure 5.1: Screenshot of the annotation tool.

and have an average age of approximately 31 years. Musicianship experience was assessed using a 5-point Likert scale where 1 means "I've never studied music theory or practice" and 5 means "I have an undergraduate degree in music". The average musicianship experience is 2.28. The participants spent on average 12 minutes and 6 seconds to annotate the two pieces.

The data collection process provides a time series of v-a values for each piece. However, only the valence dimension is needed to create a music sentiment dataset. Thus, each piece has 30 time series of valence values. The annotation of each piece was summarized into one time series and split into "phrases" of the same sentiment. Figure 5.2 illustrates how the 30 annotations of an example piece are summarized into a single time series.

The top chart shows a high inter-rater disagreement, which is explained by people's different perceptions of emotion in music. Therefore, one can't summarize the annotations by

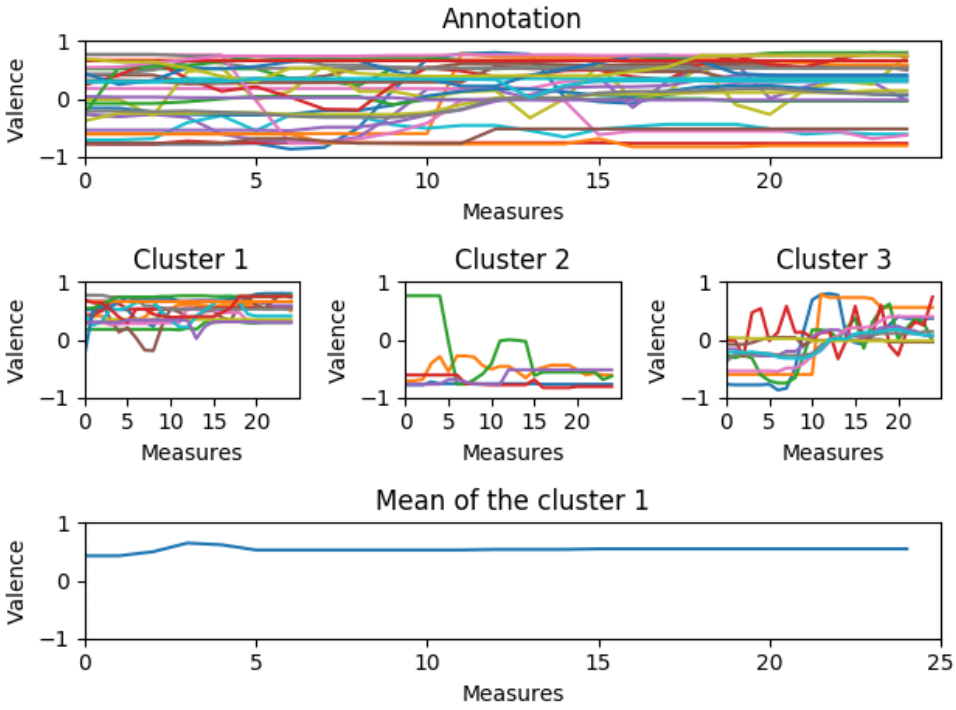


Figure 5.2: Data analysis process used to define the final label of the phrases of a piece.

directly taking their mean since that does not capture the various trends of annotation. This problem has been solved by clustering the annotations into 3 groups: *positive*, *negative*, and *ambiguous*. The *positive* cluster is created to identify the annotators that perceive the piece to be majorly positive. The *negative* cluster is intended to group the annotators that perceive the piece as mainly negative. The *ambiguous* cluster is created to group the annotators that perceive the piece to have a balanced mix of positive and negative parts. This cluster should also contain noisy annotations from participants who annotated the piece randomly to get the reward as quickly as possible. The middle charts show that clustering the annotations into 3 groups allows one to identify the annotation trends better. In this example, Cluster 1 is the

positive group, Cluster 2 is the negative group, and Cluster 3 is the ambiguous group. With these 3 clusters, the annotations are summarized by taking the mean of the cluster with the highest number of annotations. This summarizing strategy assumes that the democratic trend (defined by the majority of the annotators) defines the ground truth sentiment of the piece.

The final mean is split at all the points where the valence changes from positive to negative or vice-versa. This process creates several segments with valence values of the same sign, where segments with negative valence are considered negative phrases and segments with positive valence are positive phrases. All the phrases that had no notes (i.e., silence phrases) were removed. This process created a total of 966 phrases: 599 positive and 367 negative.

5.3 Language Model and Sentiment Classifier

Radford et al. [113] used a single-layer multiplicative LSTM (mLSTM) network [80] with 4096 neurons to learn a character-based language model from a sequence of UTF-8 encoded characters. The trained mLSTM was fine-tuned with an extra linear layer to classify sentiment in text. During fine-tuning, the pre-trained weights of the mLSTM were kept frozen, and only the weights of the extra classification layer were trained. Moreover, L1 regularization was used during fine-tuning to enforce a sparse set of weights in the classification layer. This work uses the same models and training procedures to compose music with a target sentiment.

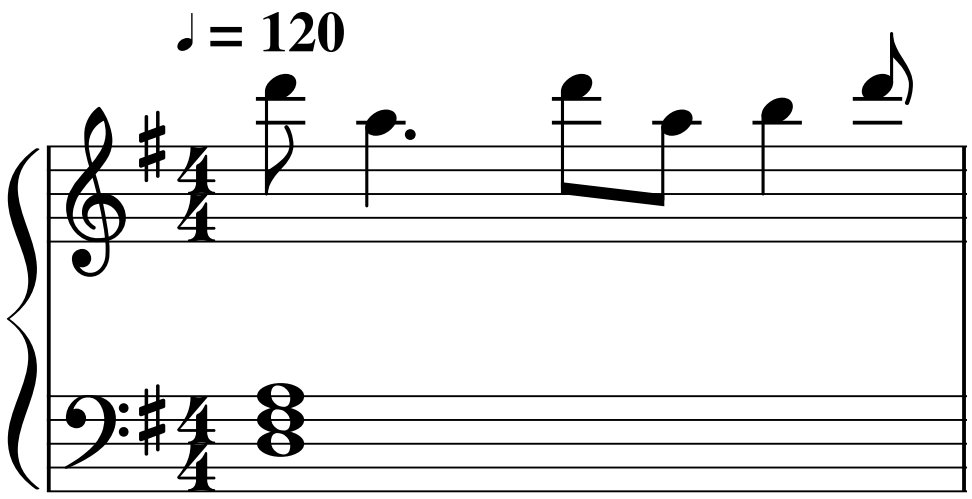
Instead of characters, this work represents music pieces as sequences of tokens from a vocabulary representing events retrieved from MIDI files. Sentiment is perceived in music due to several features such as melody, harmony, and tempo [76]. The proposed representation

attempts to encode as many music features[2] as possible while keeping the vocabulary small:

- **n_[pitch]**: play a note with a given pitch number (any integer from 0 to 127).
- **d_[duration]_[dots]**: change the duration of the following notes to a given duration with a given amount of dots. Duration types are breve, whole, half, quarter, eighth, 16th and 32nd. Dots can be any integer from 0 to 3.
- **v_[velocity]**: change the velocity of the following notes to a given velocity (loudness) number. Velocity is discretized in bins of size 4, so it can be any integer in the set $V = 4, 8, 12, \ldots, 128$.
- **t_[tempo]**: change the tempo of the piece to a given tempo in bpm. Tempo is also discretized in bins of size 4, so it can be any integer in the set $T = 24, 28, 32, \ldots, 160$.
- $\cdot$ **(dot)**: end of time step. Each time step is one sixteenth note long.
- $\backslash$**n**: end of piece.

For example, Figure 5.3 shows the encoding of the first two time steps of the first measure of the *Prelude of Light* from the *Legend of Zelda - Ocarina of Time* . The first time step sets the tempo to 120bpm, the velocity of the following notes to 76, and plays the D Major Triad for the duration of a whole note. The second time step sets the velocity to 84 and plays a dotted quarter A5 note. The total size of this vocabulary is 225, and it represents both the composition and performance elements of a piece (timing and dynamics).

[2]Constrained by the features one can extract from MIDI data.




```
t_120 v_76 d_whole_0 n_50 n_54 n_57 v_92 d_eighth n_86 . . v_84
d_quarter_1 n_81 . .
```

Figure 5.3: A short example piece encoded using the proposed representation. The encoding represents the first two time steps of the shown measure.

5.4 Empirical Evaluation

5.4.1 Sentiment Classifier

The fine-tuning approach proposed in this work is initially evaluated in the sentiment classification task. First, an mLSTM L is pre-trained as a LM with the unlabeled pieces of the VGMIDI dataset. Then, the pre-trained weights of L are frozen, and an additional linear layer E_f is stacked on top of L. The resulting model $L+E_f$ is trained as a sentiment classifier with the labeled pieces of the VGMIDI dataset. The final model $L+E_f$ is compared against a baseline mLSTM E_s trained directly on the supervised sentiment classification task. In other words, E_s is trained as a sentiment classifier only with the labeled MIDI phrases (no pre-training involved).

The unlabeled pieces used to train L are augmented in order to create additional train-

ing examples, following the methodology of Oore et al. [105]. The augmentation consists of time stretching (making each piece up to 5% faster or slower) and transposition (raising or lowering the pitch of each piece by up to a major third). All these pieces and transformations are encoded according to the proposed word-based representation (see Section 5.3). Finally, the encoded pieces are shuffled, and 90% of them are used for training and 10% for testing. The training set is divided into three shards of similar size (approximately 18,500 pieces each – 325MB), and the testing set is combined into one shard (approximately 5800 pieces – 95MB).

Six different sizes (number of neurons in the hidden layer) are compared for L: 128, 256, 512, 1024, 2048, and 4096. For each size, L is trained for 4 epochs using the 3 training shards. Weights are updated with the Adam optimizer after processing sequences of 256 words on mini-batches of size 32. The hidden and cell states of L are initialized to zero at the beginning of each shard. They are also persisted across updates to simulate full-backpropagation and allow for the forward propagation of information outside of a given sequence [113]. Each sequence is processed by an embedding layer (which is trained together with the mLSTM layer) with 64 neurons before passing through the mLSTM layer. The learning rate is initialized to $5 * 10^6$ and decayed linearly (after each epoch) to zero over the course of training.

Each L variation is evaluated with a forward pass on the test shard using mini-batches of size 32. Table 5.1 shows the average[3] cross entropy loss for each variation of L. The average cross entropy loss decreases as the size of L increases, reaching the best result (loss 1.11) when the size is equal to 4096. Thus, the variation with 4096 neurons is used to proceed with the sentiment classification experiments.

[3]Each mini-batch reports one loss.

mLSTM Size	Average Cross Entropy Loss
128	1.80
256	1.61
512	1.41
1024	1.25
2048	1.15
4096	1.11

Table 5.1: Average cross entropy loss of the mLSTM LM (L) with different size (number of neurons in the hidden layer).

After training L, an extra linear sentiment classification layer E_f is stacked on top of L and trained on the 966 labeled phrases of the VGMIDI dataset. Here, stacking means passing the final cell state of L as input to E_f. During this fine-tuning step, the base layers of L are frozen, which means only the weights of E_f are updated. Moreover, E_f is trained with L1 regularization to shrink the least important of the 4096 feature weights to zero. This ends up highlighting the neurons in L that contain most of the sentiment signal.

The fine-tuning approach $L+E_f$ was compared against the baseline supervised mLSTM E_s, which has exactly the same architecture and size as $L+E_f$ but trained in a fully supervised way (no pre-trained LM involved). The training parameters for both E_f and E_s were set to be the same used to train L. Both methods were evaluated using a 10-fold cross-validation approach, where the test folds have no phrases that appear in the training folds. Table 5.2 shows the sentiment classification accuracy of both approaches.

Method	Test Accuracy
Fine-tuned mLSTM-4096 ($L+E_f$)	89.83 ± 3.14
Baseline mLSTM-4096 (E_s)	60.35 ± 3.52

Table 5.2: Average (10-fold cross validation) sentiment classification accuracy of both fine Fine-tuned mLSTM-4096 ($L+E_f$) and Baseline mLSTM-4096 (E_s).

The fine-tuned mLSTM ($L+E_f$) achieved an accuracy of 89.83%, outperforming the baseline mLSTM (E_s) by 29.48%. The lower accuracy (60.35%) of E_s suggests that the amount of labeled data (966 phrases) was not enough to learn a good mapping between phrases and sentiment. The higher accuracy (89.83%) of $L+E_f$ shows that L is capable of learning, in an unsupervised way, a good representation of sentiment in symbolic music. This is an important result for two reasons. First, since the higher accuracy of $L+E_f$ is derived from unlabeled data, it will be easier to improve this over time using additional (less expensive) unlabeled data instead of E_s, which requires additional (expensive) labeled data. Second, because L was trained to predict the next word in a sequence, it can be used as a music generator. Since L is combined with a sentiment predictor, it opens up the possibility of generating music consistent with a desired sentiment. This idea is explored in the following section.

5.4.2 Controlling Sentiment

To control the sentiment of the music generated by the fine-tuned mLSTM ($L+E_f$), one has to find the subset of neurons in L that contain the sentiment signal by exploring the weights of the linear sentiment classification layer E_f. As shown in Figure 5.4, E_f trained with L1 regularization uses 161 neurons out of 4096. Unlike the results of Radford et al. [113], the fine-tuning step did not store the sentimental in a single neuron. Instead, the sentiment signal was stored across many neurons in a more balanced way. Therefore, one cannot simply change the values of one neuron to control the sentiment of the output music.

A genetic algorithm (GA) was used to optimize the 161 L1 weights to lead L to generate only positive or negative pieces. Each individual in the population of this GA has 161

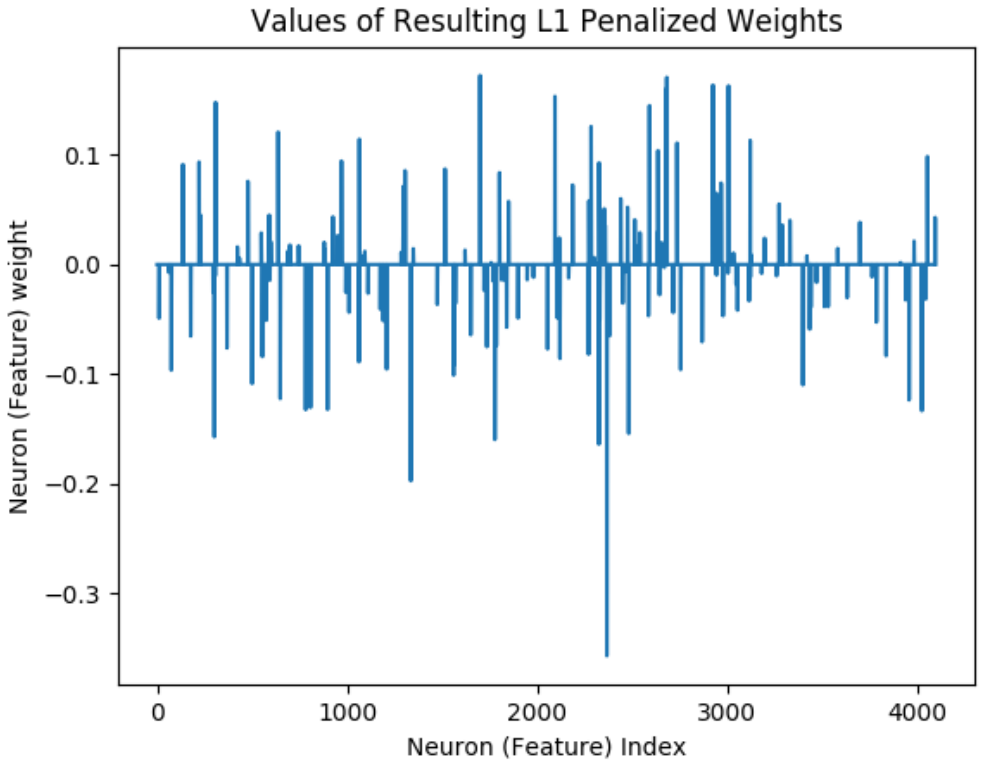


Figure 5.4: Weights of 161 L1 neurons. Note multiple prominent positive and negative neurons.

real-valued genes representing a small noise to be added to the 161 L1 weights of L. The fitness of an individual is computed by (i) adding the genes of the individual to the 161 L1 weights (vector addition) of L, (ii) generating P pieces with the updated L, (iii) using E_f to predict these P generated pieces, and (iv) calculating the mean squared error of the P predictions given a target sentiment $s \in S = \{0,1\}$. The GA starts with a random population of size 100 where each gene of each individual is a uniformly sampled random number $-2 \leq r \leq 2$. For each generation, the GA (i) evaluates the current population, (ii) selects 100 parents via a roulette wheel with elitism, (iii) recombines the parents (crossover) taking the average of their genes,

and (iv) mutates each new recombined individual (new offspring) by randomly setting each gene to a uniformly sampled random number $-2 \leq r \leq 2$.

This GA was executed twice: once to optimize L for generating positive pieces and once for negative pieces. Each execution optimized the individuals during 100 epochs with a crossover rate of 95% and a mutation rate of 10%. To calculate each individual's fitness, P=30 pieces were generated with 256 words each, starting with the symbol "." (end of time step). The optimization for positive and negative generation resulted in best individuals with fitness 0.16 and 0.33, respectively. This means that if one adds the best individual's genes of the final population to the 161 L1 weights of L, one generates positive pieces with 84% accuracy and negative pieces with 67% accuracy.

After these two optimization processes, the genes of the best final individual of the positive optimization were added to the 161 L1 weights of L. A set of 30 pieces was then generated with 1000 words starting with the symbol "." (end of time step) and 3 of them were randomly selected. The same process was repeated using the genes of the best final individual of the negative execution. Annotators were asked to label these 6 generated pieces via Amazon MTurk, using the same methodology described in Section 5.2.1. Figure 5.5 shows the average valence per measure of each of the generated pieces.

These results showed that human annotators agreed that the three positive generated pieces are indeed positive. The generated negative pieces are more ambiguous, having both negative and positive measures. However, as a whole, the negative pieces have lower valence than the positive ones. This suggests that the best negative individual (with fitness 0.33) encountered by the GA was not good enough to control the LM to generate completely negative

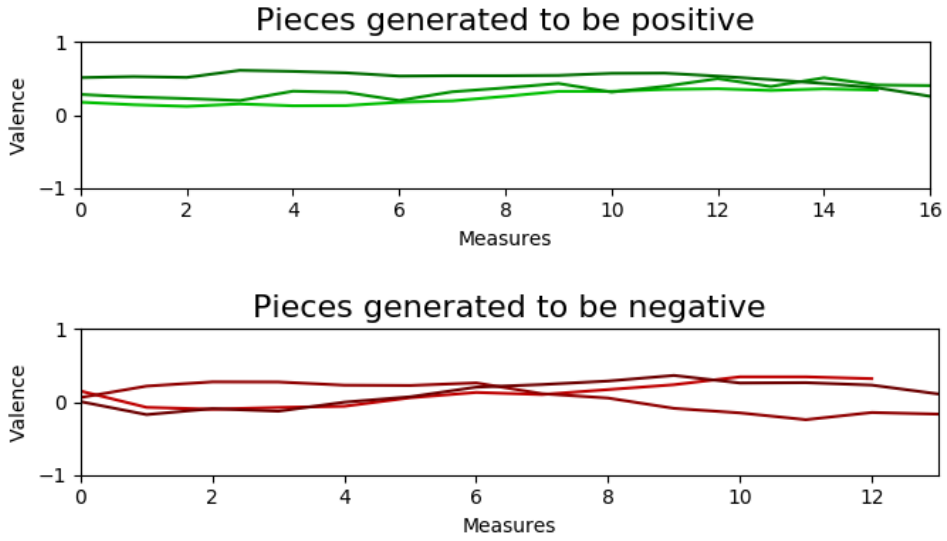


Figure 5.5: Average valence of the 6 generated pieces, as determined by human annotators. with least variance.

pieces. Moreover, the challenge to optimize the L1 weights suggests that there are more positive pieces than negative ones in the 3 unlabelled shards used to train the LM.

5.5 Conclusions

This chapter presented a method inspired by Radford et al. [113] to control a mLSTM LM to generate symbolic music with a given sentiment. The mLSTM is controlled with a genetic algorithm that optimizes the weights of specific neurons that are responsible for the sentiment signal. Such neurons are found by fine-tuning the mLSTM with an extra linear layer to classify the sentiment of symbolic music. This method of fine-tuning followed by a genetic algorithm was evaluated both as a generator and as a sentiment classifier. Results showed that the fine-tuned mLSTM obtained good classification accuracy, outperforming an equivalent mL-

STM trained in a fully supervised way. Moreover, a user study showed that humans agree that the fine-tuned LM can generate positive and negative music, with the caveat that the negative pieces are more ambiguous than the positive ones.

Chapter 6

Computer-Generated Music for Tabletop Role-Playing Games

6.1 Introduction

This chapter presents Stochastic Bi-Objective Beam Search (SBBS), a new decoding strategy inspired by Holtzman et al. [59], to generate musical pieces conveying a target emotion. SBBS works by steering at generation time the probability distribution of a LM with the probabilities given by a music emotion classifier. Unlike the method presented in the previous chapter, SBBS does not update the weights of the LM to control it towards a target emotion. SBBS is proposed as part of a system called *Bardo Composer*, or *Composer* for short, to generate background piano music for tabletop role-playing games (TRPG). Composer is applied to score sessions of Dungeons and Dragons (D&D), a TRPG in which the players interpret characters, known as player characters (PCs), in a story told by the dungeon master (DM), a special

player who also interprets all nonplayer characters (NPCs) in the story. Composers' goal is to augment the players' experience with soundtracks that match the story being told in the game. For example, if the players are fighting a dragon, Composer should generate a piece matching such a tense moment of the story. TRPG players often manually choose songs to play as background music to enhance their experience [11]. Therefore, the system should allow players to concentrate on the role-playing part of the game and not on the disruptive task of selecting the next music piece to be played.

Bardo Composer builds upon a previous system called *Bardo* [106], which selects pre-authored background music for TRPGs. Bardo uses a *naive bayes* approach to classify captions (sentences) produced by a speech recognition system into one of the four emotions: happy, calm, agitated, and suspenseful. Bardo then selects a music piece from a library corresponding to the classified emotion. The selected piece is then played as background music whenever the naive bayes classifier detects an emotion transition in the story. As shown in Figure 6.1, Composer has a similar structure as the previous system, with the major difference that Composer generates completely new pieces as opposed to select pre-authored ones. Composer also uses a speech recognition system to translate players' speeches into captions. However, unlike the previous system, Composer classifies the captions with a transformer according to a discrete circumplex model of emotion. A second transformer is trained to classify the emotion of symbolic piano pieces according to the same discrete circumplex model. Composer then uses SBBS to control a LM with the music emotion classifier to generate music pieces conveying the target emotion given by the story emotion classifier.

Two new datasets have been created to support both Bardo and Composer [106, 41]:

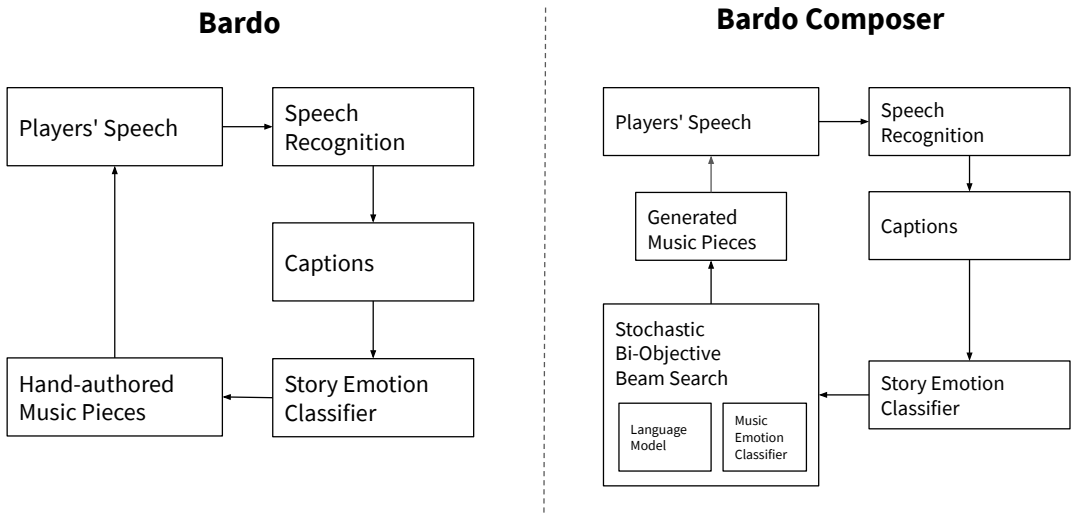


Figure 6.1: Diagrams with the architecture of Bardo (left) and Composer (right).

Call of the Wild (CotW) and *ADL Piano Midi*. CotW is a dataset of transcribed captions from YouTube videos of a D&D campaign. ADL Piano Midi is a large and diverse MIDI dataset with piano pieces extracted from the Lakh MIDI dataset [116]. Moreover, in this work, the VGMIDI dataset has been extended from 95 to 200 labeled pieces using the same annotation method as the original dataset (see Chapter 5). All the 200 pieces are piano arrangements of video game soundtracks labeled according to the circumplex model of emotion. The discrete circumplex model of emotion used by Composer is defined to integrate emotions of the CotW stories with the VGMIDI pieces.

A listening test with 116 participants was performed to evaluate whether human subjects are able to correctly identify the emotion conveyed in pieces generated by Composer. Results showed that human subjects could correctly identify the emotion of the generated pieces as accurately as they were able to identify the emotion of pieces written by humans. The original

Bardo was published in the Proceedings of the 13th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE17) [106] and Bardo Composer in the Proceedings of AIIDE20 [41].

6.2 Datasets

6.2.1 Call of the Wild

A new dataset called *Call of the Wild* (CotW) was created to train the story emotion classifier of the original Bardo. This dataset includes 9 episodes of the CotW D&D campaign available on YouTube, which sum a total of 4 hours, 39 minutes, and 24 seconds of gameplay. Each episode is an independent YouTube video with approximately 30 minutes played by the same 4 players (1 DM and 3 PCs). All videos were processed by the YouTube speech recognition system that generated the English captions from the speeches of the 4 players. This process yielded 5,892 sentences (45,247 words), which 3 different annotators labeled according to a categorical model with four emotions: happy, calm, agitated, and suspenseful. The annotation process assumed the PCs perspective in the story, as there could be moments that PCs and NPCs could experience different emotions. Should two annotators agree on the label of a sentence s, then s is labeled according to the two annotators. One of the annotators watched the videos again to break the ties (sentences that each annotator attributed a distinct label for).

Composer's story emotion classifier is also trained with the CotW dataset. However, in order to have an integrated model of emotion between the CotW stories and the VGMIDI music pieces, Composer uses a discretized circumplex model of emotion [121] that generalizes

the four emotion model used in Bardo [106]. In Composer's emotion model, the dimensions of valence and arousal assume binary values $v \in [-1, 1]$ and $a \in [-1, 1]$, respectively. Valence measures sentiment and thus $v = -1$ means a negative emotion and $v = 1$ means a positive emotion. Arousal measures the energy of the emotion, and thus $a = -1$ means that the emotion has low energy, whereas $a = 1$ means that the emotion has high energy. With this discretized circumplex model, Bardo's four emotion model is mapped to the following emotions:

1. Suspenseful is mapped to low valence and arousal ($v = -1, a = -1$).
2. Agitated is mapped to low valence and high arousal ($v = -1, a = 1$).
3. Calm is mapped to high valence and low arousal ($v = 1, a = -1$).
4. Happy is mapped to high valence and arousal ($v = 1, a = 1$).

This mapping is based on the circumplex model used to annotate the VGMIDI dataset. When human subjects annotated that dataset, they used a continuous circumplex model with labels defining a fixed set of discrete basic emotions (see Figure 5.1). This mapping allows one to use the circumplex model with the labeled CotW dataset. For example, in the context of D&D, the sentence "Roll initiative" is normally said at the beginning of battles and it can be considered ($v = -1, a = 1$), once a battle is a negative (dangerous) moment with high energy. "Roll initiative" is normally classified as agitated in the CotW dataset [106].

6.2.2 ADL Piano MIDI

Composer trains its LM with a new dataset called ADL (Augmented Design Lab) Piano MIDI. This new dataset is based on the Lakh MIDI dataset [116], which is one of the

largest MIDI collections publicly available. The Lakh MIDI dataset contains 45,129 unique MIDI files in which different versions of the same piece may occur. Only one version of each piece was kept. Given that Composer focuses on piano pieces, only the tracks with instruments from the piano family (MIDI program numbers 1-8 in the dataset) were considered from the Lakh MIDI dataset. This process yielded a total of 9,021 unique piano MIDI files. These files are mainly Rock and Classical pieces, so to increase genre diversity in the dataset (e.g., Jazz, Blues, and Latin), an additional 2,065 files were included from public sources on the Internet[1]. All files in the final collection were de-duped according to their MD5 checksum. The final dataset has 11,086 unlabeled MIDI piano pieces. The motivation to create this new dataset instead of using the unlabelled VGMIDI pieces is to have a considerably larger and varied symbolic music dataset to support better pre-trained LMs.

6.3 Bardo Composer

The general structure of Composer is formalized in Algorithm 2. It receives as input a speech recognition system S, a story emotion classifier E_s, a music emotion classifier E_m, a LM for symbolic music generation L, a speech signal p with the last sentences spoken by the players, and a sequence x of musical symbols composed in previous calls to Composer. The algorithm also receives parameters b and k, which are used in `SBBS` described in Algorithm 3. Composer returns a symbolic piece that tries to match the emotion in the players' speeches. Composer starts by transcribing the speech signal p into a caption s with S (line 1). In addition to a caption, S returns the duration l of the signal p in seconds. Then, Composer classifies the

[1] `https://bushgrafts.com/midi/` and `http://midkar.com/jazz/`

emotion of s in terms of valence v and arousal a and it invokes SBBS to generate a sequence of symbols y that matches the desired length l and emotion with arousal a and valence v. SBBS receives as input the models L and E_m, the current sequence x, the desired emotion values v and a, SBBS's parameter values b and k, and the desired length l of the piece to be generated.

Algorithm 2 Bardo Composer

Require: Speech recognition system S, Text emotion classifier E_s, Music emotion classifier E_m, LM L, speech signal p, previously composed symbols x, beam size b, number of symbols k

Ensure: Music piece x

1: $s, l \leftarrow S(p)$

2: $v, a \leftarrow E_s(s)$

3: $y \leftarrow \texttt{SBBS}(L, E_m, x, v, a, b, k, l)$ # *see Algorithm 3*

4: **return** $x \cup y$

In the first call to Composer, the sequence x is initialized with the symbols of the first 4 timesteps of a random human-composed piece with the emotion v, a, as returned by E_s. Every time there is a transition from one emotion to another, the sequence x is reinitialized using the same process. This is used to bias the generative process and to emphasize emotion transitions. To be used in real-time, Composer is invoked with the most recently captured speech signal p and returns a composed piece of music. While the most recent piece is being played at the game table, Composer receives another signal p and composes the next excerpt. One also needs to define the length of the signal p. Similar to Bardo [106], Composer uses YouTube's subtitle system as the speech recognition system S. Therefore, signals p are long enough to form a sentence in the form of a caption.

6.3.1 Story Emotion Classifier

Composer's story emotion classifier E_s is trained with a transfer learning approach, in which a pre-trained BERT_{BASE} LM [25] is fine-tuned with the CotW dataset. This transfer learning approach is used because the pre-trained BERT_{BASE} LM has been shown to boost models' performance on a wide range of NLP tasks [25]. BERT_{BASE} uses transformer encoder blocks with 12 layers, 768 units per layer, and 12 attention heads. It was pre-trained with both the BooksCorpus dataset (800M words) [154] and the English Wikipedia (2,500M words). Although in Algorithm 2 the story emotion classifier is depicted as a single model E_s, in practice, Composer treats valence and arousal independently. Thus, BERT_{BASE} is independently fine-tuned on the CotW dataset [106] for each dimension of the circumplex model. Fine-tuning consists of adding a linear classification head on top of the pre-trained BERT_{BASE} and training all layers (including the pre-trained ones) of the resulting model end-to-end.

6.3.2 Language Model

At the time Composer was proposed, there was no publicly available high-capacity LM pre-trained with large MIDI datasets for symbolic music modeling. Therefore, a high-capacity GPT-2 LM was pre-trained on the ADL Piano MIDI dataset. The GPT-2 LM has 4 layers (transformer blocks), a context size of 1024 tokens, 512 embedding units, 1024 hidden units, and 8 attention heads. Composer uses GPT-2 instead of BERT because GPT-2 is better suited for sequence generation than BERT.

To process MIDI files as sequences with the GPT-2 LM, Composer encodes a MIDI file by parsing all notes from the `NOTE_ON` and `NOTE_OFF` events in the MIDI. A note is defined

as a set $z = (z_p, z_s, z_d, z_v)$, where $\{z_p \in \mathbb{Z} | 0 \leq z_p < 128\}$ is the pitch number, $\{z_s \in \mathbb{Z} | z_s \geq 0\}$ is the note starting time in timesteps, $\{z_d \in \mathbb{Z} | 0 \leq z_d \leq 56\}$ is note duration in timesteps and $\{z_v \in \mathbb{Z} | 0 \leq z_v < 128\}$ is the note velocity. Given a MIDI `NOTE_ON` event, a note z is parsed by retrieving the starting time z_s (in seconds), the pitch number z_p and the velocity z_v from that event. To calculate the note duration z_d, the end time z_e (in seconds) of the corresponding `NOTE_OFF` event is retrieved. Thus, the note duration $z_d = \lfloor t \cdot z_e \rfloor - \lfloor t \cdot z_s \rfloor$ is computed from the discretized durations z_s and z_e, where t is a parameter defining the sampling frequency of the timesteps. Composer derives a sequence $x = \{z_v^1, z_d^1, z_p^1, \cdots, z_v^n, z_d^n, z_p^n\}$ of tokens for a given MIDI file by (a) parsing all notes z^i from the file, (b) sorting them by starting time z_s^j, and (c) concatenating their velocity z_v^j, duration z_d^j, and pitch z_p^j. Composer adds two special tokens `TS` and `END` in the sequence x, to mark the end of a timestep and the end of a piece, respectively.

This encoding yields a vocabulary V of size $|V| = 314$, which is slightly greater than the size of the vocabulary used in Chapter 5 (225 tokens). This increase comes from the extra number of note durations that can be represented in this new scheme. The main difference between these two schemes is that the new one does not encode tempo directly. The tempo of the pieces is implicitly encoded in the duration of the notes. Faster pieces have shorter notes, whereas slower pieces have longer notes. Moreover, the duration of the notes is not encoded as a duration type, as in Chapter 5. In this new scheme, the duration is represented as the number of time steps that note lasts. The main motivation for this new scheme is to provide a more accurate representation of note durations without considerably increasing the size of the vocabulary.

6.3.3 Music Emotion Classifier

As was the case with E_s, the emotion music classifier E_m is also trained with a transfer learning approach. E_m fine-tunes the GPT-2 LM pre-trained on the ADL Piano MIDI dataset. Similar to E_s, E_m also treats valence and arousal independently. Thus, the pre-trained GPT-2 is fine-tuned on the extended VGMIDI dataset for each dimension of the circumplex model. Following the approach of Radford et al. [114], these models were fine-tuned by adding and extra layer to the pre-trained LM and training the entire model (including the pre-trained layers) with the VGMIDI dataset [42].

6.3.4 Stochastic Bi-Objective Beam Search

Composer uses a new search-based decoding algorithm called *Stochastic Bi-Objective Beam Search* (`SBBS`), which combines a LM and a music emotion classifier to bias the process of music generation to match a target emotion (line 3 of Algorithm 2). Given a LM L and the music emotion classifiers $E_{m,v}$ and $E_{m,a}$, for valence and arousal, respectively, the goal of `SBBS` is to allow for the generation of pieces that sound "good" (i.e., have high probability value according to the trained LM), but that also match the current emotion of the story being told by the players. `SBBS` is stochastic because it samples from a distribution instead of greedily selecting the best sequences of symbols, as a regular beam search does. The regular beam search is deterministic and so it always generate the same piece for a given input sequence x and emotion (v,a). The stochasticity of `SBBS` allows it to generate different music pieces given the same input parameters x and (v,a). `SBBS` is "bi-objective" because it optimizes for realism and emotion. Algorithm 3 shows `SBBS`'s pseudocode. The letters x, y, and m denote sequences

of musical symbols. Function $p_L(y) = \prod_{y_t \in y} P(y_t | y_1, \cdots, y_{t-1})$ is the probability of sequence y according to the LM L; a high value of $p_L(y)$ means that y is recognized as a piece of "good quality" by L. Function $l(y)$ denotes the duration in seconds of piece y. Finally, $x[i:j]$ denotes the subsequence of x starting at index i and finishing at index j.

Algorithm 3 Stochastic Bi-Objective Beam Search

Require: Music emotion classifier E_m, LM L, previously composed symbols x, valence and arousal values v and a, number k of symbols to consider, beam size b, length l in seconds of the generated piece.

Ensure: Sequence of symbols of l seconds.

$B \leftarrow [x]$, $j \leftarrow 0$
while $l(y[t:t+j]) < l, \forall y \in B$ **do**
  $C \leftarrow \{\}$
  for all $m \in B$ **do**
    $C_m \leftarrow \{m \cup s | s \in V\}$
    $C_k \leftarrow k$ elements y from C_m with largest $p_L(y)$
    $C \leftarrow C \cup C_k$
  end for
  $B \leftarrow b$ sequences y sampled from C proportionally to $p_L(y)(1 - |v - E_{m,v}(y)|)(1 - |a - E_{m,a}(y)|)$
  $j \leftarrow j + 1$
end while
return $m \in B$ such that $p_L(m) = \max_{y \in B} p_L(y)$ and $l(y[t:t+j]) \geq l$

SBBS initializes the beam structure B with the sequence x passed as input (line 1). SBBS also initializes variable j for counting the number of symbols added by the search. SBBS keeps in memory at most b sequences and, while all sequences are shorter than the desired duration l (line 2), it adds a symbol to each sequence (lines 3–10). In line 3, SBBS initializes a set C to store the candidate solutions of the next beam structure B. SBBS then generates all sequences by adding one symbol from vocabulary V to each sequence m from B (line 5). These extended sequences, known as the children of m, are stored in C_m. The operations performed in lines 6 and 9 attempt to ensure the generation of good pieces that convey the desired emotion. In line 6, SBBS selects the k sequences with the largest p_L-values among the children of m. This is because some of the children with low p_L-value could be attractive from the perspective of the desired emotion and, although the resulting piece could convey the desired emotion, the piece would be of low quality according to the LM. The best k children of each sequence in the beam are added to set C (line 7). Then, in line 9, SBBS samples the sequences that will form the beam of the next iteration. Sampling occurs proportionally to the values of $p_L(y)(1-|v-E_{m,v}(y)|)(1-|a-E_{m,a}(y)|)$, for sequences y in C. A sequence y has higher chance of being selected if L attributes a high probability value to y and if the music emotion model classifies the values of valence and arousal of y to be similar to the desired emotion. When at least one of the sequences is longer than the desired duration of the piece, SBBS returns the sequence with largest p_L-value that satisfies the duration constraint (line 12).

6.4 Empirical Evaluation

SBBS is empirically evaluated with two experiments. The first one evaluates the accuracy of the models for story and music emotion classification. The fine-tuned BERT_{BASE} model for story emotion classification is compared against the simpler Naïve Bayes approach used in the original Bardo [106]. The fine-tuned GPT-2 model for music emotion classification is compared against the simpler fine-tuned mLSTM presented in Chapter 5. The second experiment evaluates, with a listening test, whether human subjects can recognize different emotions in pieces generated by Composer for the CotW campaign.

6.4.1 Emotion Classifiers

6.4.1.1 Story Emotion

The story emotion classifier is a pair of BERT_{BASE} models, one for valence and one for arousal. Both these models were fine-tuned for 10 epochs with the Adam optimizer [77]. Each training step was performed over a mini-batch of size 32. The learning rate was set to 3e-5 and the dropout rate to 0.5. The CotW dataset is divided into 9 episodes. Thus, the accuracy of each BERT_{BASE} classifier is evaluated with a leave-one-out strategy. Given a set of episodes E, for each episode $e \in E$, the $E - e$ episodes are used for training, and the episode e is used for testing. For example, when testing on episode 1, episodes 2-8 are used for training. Every sentence is encoded using a WordPiece embedding [144] with a 30,000 token vocabulary.

The fine-tuned BERT_{BASE} classifiers are compared with Bardo's Naive Bayes (NB) approach, which encodes sequences using bag-of-words with tfidf. Table 6.1 shows the accu-

racy of the valence classification of both these methods per episode. The BERT$_{BASE}$ classifier outperforms NB in all the episodes, having an average accuracy 7% higher. For valence classification, the hardest episode for both the models is episode 7, where BERT$_{BASE}$ had the best performance improvement when compared to NB. Episode 7 is different from all other episodes. While the other episodes are full of battles and ability checks, episode 7 is mostly PCs talking with NPCs. Therefore, what is learned in the other episodes does not generalize well to episode 7. The improvement in accuracy of BERT$_{BASE}$ in that episode is likely due to the model's pre-training. Episodes 5 and 9 were equally easy for both methods because these episodes are similar to one another. What is learned in one of these episodes generalizes well to the other.

Alg.	**Episodes**									**Avg.**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
NB	73	88	91	85	94	81	41	74	94	80
BERT$_{BASE}$	**89**	**92**	**96**	**88**	**97**	**81**	**66**	**83**	**96**	**87**

Table 6.1: Valence accuracy in percentage of Naive Bayes (NB) and BERT$_{BASE}$ for story emotion classification.

Table 6.2 shows the accuracy of arousal classification of both NB and BERT$_{BASE}$. Again BERT$_{BASE}$ outperforms NB in all the episodes, having an average accuracy 5% higher. In contrast with the valence results, here there is no episode in which BERT$_{BASE}$ substantially outperforms NB.

6.4.1.2 Music Emotion

The music emotion classifier is a pair of GPT-2 models, one for valence and one for arousal. First, a GPT-2 LM is pre-trained with the new ADL Piano MIDI dataset. Each piece

Alg.	Episodes									Avg.
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
NB	82	88	75	79	82	76	98	86	84	83
BERT	**86**	**90**	**77**	**86**	**89**	**88**	**99**	**90**	**88**	**88**

Table 6.2: Arousal accuracy in percentage of Naive Bayes (NB) and BERT$_{BASE}$ for story emotion classification.

p of this dataset is augmented by (a) transposing p to every key, (b) increasing and decreasing p's tempo by 10%, and (c) increasing and decreasing the velocity of all notes in p by 10% [105]. Thus, each piece generated $12 \cdot 3 \cdot 3 = 108$ different examples. The pre-trained GPT-2 LM is then fine-tuned twice on the VGMIDI dataset, once for valence and once for arousal. It is important to highlight that the pieces used to pre-train the LM can be safely augmented because they are unlabelled. The labeled pieces are not augmented because that could affect the valence or arousal of the pieces.

Similar to the story emotion classifiers, the GPT-2 classifiers are fine-tuned for 10 epochs using an Adam optimizer with a learning rate of 3e-5. Unlike the story emotion classifiers, each training step is performed over mini-batches of size 16 (due to GPU memory constraints) and dropout of 0.25. The VGMIDI dataset is defined with training and testing splits of 160 and 40 pieces, respectively. Each piece p of this dataset was augmented by slicing p into 2, 4, 8, and 16 parts of equal length and emotion. Thus, each part of each slicing generated one extra example. This augmentation is intended to help the classifiers generalize for pieces with different lengths.

The fine-tuned GPT-2 classifiers are compared with mLSTM models that are also pre-trained with the ADL Piano Midi dataset and fine-tuned with the VGMIDI dataset. These base-

lines were chosen because they are currently the state-of-the-art models in the VGMIDI dataset [42]. The mLSTMs have the same size as the GPT-2 models (4 hidden layers, 512 embedding units, 1024 hidden units) and are pre-trained and fine-tuned with the same hyper-parameters. Table 6.3 shows the accuracy of both models for valence and arousal. The performance of these models without pre-training (i.e., trained only on the VGMIDI dataset) is also reported. These are the baseline versions of the models.

Algorithm	**Valence**	**Arousal**
Baseline mLSTM	69	67
Fine-tuned mLSTM	74	79
Baseline GPT-2	70	76
Fine-tuned GPT-2	**80**	**82**

Table 6.3: Accuracy in percentage of both the GPT-2 and mLSTM models for music emotion classification.

Results show that using transfer learning can substantially boost the performance of both the GPT-2 and the mLSTM. The fine-tuned GPT-2 is 10% more accurate than its respective baseline in terms of valence and 8% in terms of arousal. The fine-tuned mLSTM is 5% more accurate than its respective baseline in terms of valence and 12% in terms of arousal. Finally, the fine-tuned GPT-2 outperformed the fine-tuned LSTM by 6% and 3% in terms of valence and arousal, respectively.

6.4.2 Listening Test

Composer's performance in generating music that matches the emotions of a story is assessed with a listening test. Composer is applied to generate a piece for a snippet composed of 8 contiguous sentences of each of the first 5 episodes of the CotW dataset. Each snippet has

one emotion transition that happens in between sentences. The sentences are 5.18 seconds long on average. To test Composer's ability to generate music pieces with emotion changes, human subjects were asked to listen to the 5 generated pieces and evaluate the transitions of emotion in each generated piece. The listening test was performed via Amazon Mechanical Turk and had an expected completion time of approximately 10 minutes. A reward of USD $1 was given to each participant who completed the study.

In the first section of the study, the participants were presented with an illustrated description of the circumplex model of emotion and listened to 4 examples of labeled pieces from the VGMIDI dataset. Each piece had a different emotion: low valence and arousal, low valence and high arousal, high valence and low arousal, high valence and arousal. In the second section of the study, participants were asked to listen to the 5 generated pieces (one per episode). After listening to each piece, participants had to answer 2 questions: (a) "What emotion do you perceive in the 1st part of the piece?" and (b) "What emotion do you perceive in the 2nd part of the piece?" To answer these two questions, participants selected one of the four emotions: low valence and arousal, low valence and high arousal, high valence and low arousal, high valence and arousal. Subjects were allowed to play the pieces as many times as they wanted before answering the questions. The final section of the study was a demographics questionnaire including ethnicity, first language, age, gender, and experience as a musician. To answer the experience as a musician, the participants used a 5-point Likert scale where 1 means "I've never studied music theory or practice" and 5 means "I have an undergraduate degree in music".

Composer is compared with a baseline method that selects a random piece from the VGMIDI dataset whenever there is a transition of emotion. The selected piece has the same

emotion of the sentence (as given by the story emotion classifier). A between-subject strategy was used to compare these two methods, where Group *A* of 58 participants evaluated the 5 pieces generated by Composer and Group *B* of 58 participants evaluated the 5 pieces from the baseline. This strategy was used to avoid possible learning effects where subjects could learn emotion transitions from one method and apply the same evaluation directly to the other method. The average age of groups *A* and *B* are 34.96 and 36.98 years, respectively. In Group *A*, 69.5% of the participants are male and 30.5% are female. In Group *B*, 67.2% are male and 32.8% are female. The average musicianship of the groups *A* and *B* are 2.77 and 2.46, respectively.

Table 6.4 shows the results of the listening test. The two parts (p1 and p2 in the table) of each episode are considered independent pieces. The table presents the percentage of participants that correctly identified the pieces' valence and arousal (*v* and *a* in the table, respectively), as intended by the methods. This percentage is refeered to as the approach's accuracy. For example, 87% of the participants correctly identified the arousal value that Composer intended the generated piece for part p1 of episode 4 (e4-p1) to have. The approaches' average accuracy is also presented across all pieces (*Average* in the table) in terms of valence, arousal, and jointly for valence and arousal (*va* in the table). The *va*-value of 34 for Composer means that 34% of the participants correctly identified the system's intended values for valence and arousal across all pieces generated.

At first, one might expect the baseline method to get close to 100% accuracy, especially in the first parts of the episodes, since the baseline selects human-composed pieces from the VGMIDI dataset. However, the results showed the best accuracy value for baseline is 79% on e2-p2. These low accuracy values of baseline suggest that the evaluation methodology used

Method	Episodes																				Average		
	e1-p1		e1-p2		e2-p1		e2-p2		e3-p1		e3-p2		e4-p1		e4-p2		e5-p1		e5-p2				
	v	a	v	a	v	a	v	a	v	a	v	a	v	a	v	a	v	a	v	a	v	a	va
Baseline	56	**65**	39	56	39	62	39	**79**	**48**	**60**	**67**	**53**	**58**	70	**63**	**75**	25	36	**72**	58	**51**	**32**	**34**
Composer	**62**	60	**44**	**65**	**82**	**68**	**53**	68	24	55	46	43	25	**87**	37	55	**81**	**86**	51	**67**	**51**	30	**34**

Table 6.4: The percentage of participants that correctly identified the valence and arousal (v and a, respectively) intended by the methods for the pieces parts (p1 and p2).

in this study is significantly different from the annotation process used to annotate the VGMIDI dataset. Typically, people use words to describe perceived emotions in music, but the circumplex model requires them to use pairs of valence-arousal values. The annotation task of the VGMIDI dataset is slightly easier because the annotators can guide their decision with the basic emotions (e.g., happy, sad, etc.) labeled on the model. In this new experiment, the participants didn't have this visual help. This difference between the annotation tasks can explain the disagreement of the participants of this new experiment with the annotators of the VGMIDI dataset.

Comparing the accuracies of the two methods, Composer outperformed the Baseline in e1-p2, e2-p1, and e5-p1. Baseline outperformed Composer in e3-p1, e3-p2 and e4-p2. In the other four parts, one method performed better for valence, whereas the other performed better for arousal. Surprisingly, Composer (generative approach) outperformed the Baseline (human-composed pieces) in 3 parts (e1-p2, e2-p1, and e5-p1). These results can be explained by the fact that Composer starts a piece with the first 4 timesteps (4 seconds) of a VGMIDI piece selected randomly from the subset of pieces with the target emotion. Thus, the introduction of a Composer part, which arguably is the most important aspect to define the perception of an emotion transition, is also coming from a human-composed piece. However, each episode part

in this experiment has an average length of 20.72 seconds. The surprisingly positive results of Composer suggest that SBBS can develop the human-composed introduction into pieces that also represent the target emotion, which can be attributed to high-accuracies for valence and arousal of the music emotion classifier. An even more surprising result is the accuracy of Composer in e5-p1, which is much higher than the Baseline. Since the introduction of e5-p1 is not the same between the two systems, this result suggests that the introduction randomly selected for Composer represents better the emotion of e5-p1 than the introduction in the piece chosen for the Baseline.

Overall, the average results show that both systems performed very similarly. Both of them had an average accuracy on the combined dimensions equal to 34%. The difference between these two methods and a system that selects pieces at random (expected accuracy of 25%) is significant according to a Binomial test ($p = 0.02$). These results show that the participants were able to identify the emotions in the generated pieces as accurately as they were able to identify the emotions in human-composed pieces. This is an important result towards the development of a fully automated system for music composition for story-based tabletop games.

6.5 Conclusions

This chapter presented SBBS, a new search-based decoding algorithm to compose music with a target emotion. SBBS is presented as part of Bardo Composer, a system that automatically composes music for tabletop role-playing games. The system processes sequences

from speech and generates pieces one sentence after the other. The emotion of the sentence is classified using a fine-tuned BERT model. This emotion is given as input to SBBS, which generates a piece that matches the emotion with the guidance of a fine-tuned GPT-2 music emotion classifier.

Bardo Composer was evaluated with a listening test, in which human subjects were asked to evaluate the transitions of emotion in pieces generated for different episodes of a D&D game. Bardo Composer was compared against a baseline method that randomly selects pieces from the VGMIDI dataset whenever there is a transition of emotion in the game's story. Results showed that human subjects correctly identified the emotion of the generated music pieces as accurately as they could identify the emotion of pieces composed by humans. This surprisingly positive result can be explained by the good performance of the GPT-2 music emotion classifier, which allows SBBS to keep the intended emotion when generating pieces conditioned by the beginning of pieces from the VGMIDI dataset.

Chapter 7

Controlling Emotions in Symbolic Music Generation with MCTS

7.1 Introduction

This chapter presents a new decoding strategy based on Monte Carlo Tree Search (MCTS) to generate musical pieces conveying a target emotion. Similar to SBBS, MCTS uses a music emotion classifier E to steer the probability distribution of a neural LM L towards a target emotion e at generation time. Unlike SBBS, MCTS performs multiple search iterations for each decoded token. Each iteration uses the Predictor Upper Confidence for Trees (PUCT) to update the distribution of node visits N, where E determines the expected reward (*Q value*) of each node and L the prior probability of selecting each node (*P value*). After all the search iterations to decode the next token, N ends up steering L towards e. Therefore, the next token is decided by sampling from N.

The search is performed over the space of sequences learned by a music transformer LM [63] with the unlabelled pieces of the VGMIDI dataset. The number of unlabelled pieces of the VGMIDI dataset has been extended as part of this work from 728 to 3,640. Similar to the approaches presented in Chapter 5 [42] and Chapter 6 [41], the music emotion classifier is trained by fine-tuning a pre-trained LM with a classification head on the 200 labeled pieces of the VGMIDI dataset. Unlike these two previous approaches, emotion classification is treated as a multiclass problem instead of a binary one. In this new framing, each of the four quadrants of the circumplex model is mapped to a label.

MCTS is evaluated with two listening tests, one to measure the quality of the generated pieces and one to measure MCTS's accuracy in generating pieces with a target emotion. In the first test, human subjects are asked to prefer between MCTS, the validation data (human compositions), TopK sampling, and SBBS [41]. In the second test, human subjects are asked to annotate the emotions they perceive in pieces generated by MCTS, TopK sampling, and SBBS. Results show that MCTS outperforms SBBS in terms of music quality while slightly improving the accuracy of conveying target emotions. MCTS and TopK performed similarly in terms of music quality. An expressivity analysis [127] is performed to evaluate how MCTS conveys each target emotion. The frequencies of pitch classes and note durations suggest that MCTS can reproduce some common composition practices used by human composers.

7.2 Language Model

Music transformer [63] is currently one of the state-of-the-art NN architectures for symbolic music generation, and hence it is used to train a base LM for MCTS. This LM is trained on the unlabelled pieces of the VGMIDI dataset. Originally, the VGMIDI dataset had 728 unlabelled pieces, but it has been expanded to 3,640 pieces in this work. The new pieces are piano arrangements of video game music created by the NinSheetMusic community[1]. The VGMIDI dataset has been used, instead of other large datasets of symbolic music (e.g., the MAESTRO dataset [53]), to be able to train both the LM and the music emotion classifier on similar datasets.

The VGMIDI pieces are encoded using a different vocabulary than the original one proposed with music transformer [63]. Aiming at reducing the length of the music sequences, the MIDI files are mapped to sequences using a large expressive vocabulary instead of a compact one. To create a sequence from a MIDI file, the starting times of all the notes are discretized into a sequence of time steps. Then, each time step is processed in order, generating a token $n_{p,d,v}$ for each note in the time step. The three parameters of a note token $n_{p,d,v}$ are pitch p, duration d and velocity v, respectively. In order to constrain the possible combinations of note tokens, the pitch values are limited to $30 \leq p \leq 96$. Duration d is limited by the types: breve, whole, half, quarter, eighth, 16th and 32nd. The dotted versions of these types (maximum of 3 dots) are also considered. Velocities are limited to the values $v \in [32, 36, 40, \cdots, 128]$. After processing each time step, a token r_d is generated representing a rest with a given duration d. The token "." (period) is included at the end of all time steps to represent the end of the piece.

[1] https://www.ninsheetmusic.org/

This encoding scheme yields a vocabulary with 44,346 tokens, which is orders of magnitude larger than the vocabularies described in Chapters 5 (225 tokens) and 6 (314 tokens). The benefit of this large vocabulary over the small ones described previously is that it allows the MIDI files to be mapped to considerably smaller sequences. Music transformer networks can only process sequences with a fixed size. Thus, reducing the size of the encoded pieces allows the music transformer to model pieces entirely and hence capture long-term dependencies in music. Moreover, according to Holtzman et al. [59], LMs with larger vocabularies tend to generate less repetitive sequences. On the other hand, by having more tokens, MCTS has more options to choose from at any given point of the generative process, which increases the search complexity. MCTS mitigates the effects of having more options to choose from by only considering the top k tokens according to the LM at any decision point of the generative process.

7.3 Music Emotion Classifier

In chapters 5 and 6, it has been shown that fine-tuning a LM with an extra classification head yields a better model than training a classifier from scratch with the same architecture of the LM. This work follows a similar approach. The Music Transformer LM described in the previous section is fine-tuned as a music emotion classifier with the labeled pieces of the VGMIDI dataset. In chapters 5 and 6, symbolic music emotion classification is approached as two independent binary problems, one for valence and one for arousal. This work defines it as a multiclass problem. Four emotions are considered: high valence and arousal (E0), low valence and high arousal (E1), low valence and arousal (E2), and high valence and low arousal (E4).

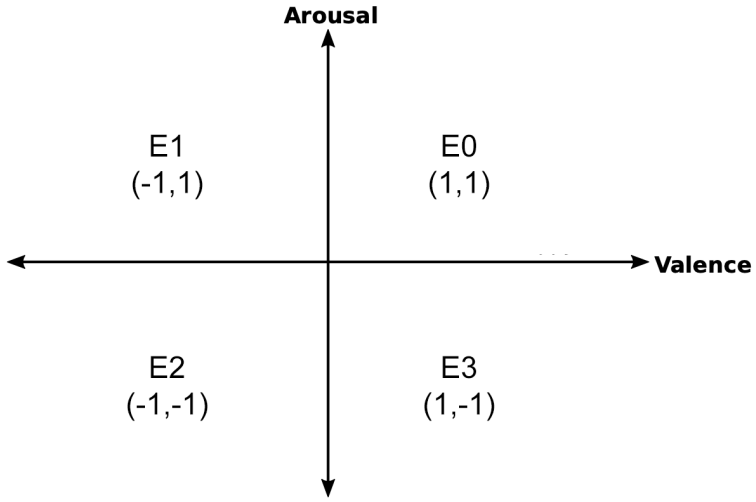


Figure 7.1: Mapping the circumplex model to a categorical model of emotion with four classes: E0, E1, E2, and E3.

Each labeled piece in the VGMIDI dataset has a valence label $v \in [-1,1]$ and an arousal label $a \in [-1,1]$. A pair of values is mapped to a categorical label (E0, E1, E2, or E3) by getting the quadrant of (v,a). As shown in Figure 7.1, a piece with values $(1,1)$ is mapped to E0, $(-1,1)$ to E1, $(-1,-1)$ to E2, and $(1,-1)$ to E3.

This mapping yields 76 pieces with label E0, 38 with label E1, 27 with label E2, and 59 with E3. This multiclass approach is used to simplify the search task of controlling the emotion of the generated pieces: only a single model defines the emotion scores instead of a combination of two models. Since MCTS evaluates the emotion of sequences with different lengths (see Section 7.4), the music emotion classifier is trained with prefixes of different lengths extracted from the labeled pieces. Thus, during training, the classifier learns a mapping of a sequence to an emotion considering different lengths. This is especially helpful to guide the search at the beginning of the generation process, where the number of generated tokens is short,

and so the emotion classifier does not have much context to be confident about a prediction.

7.4 MCTS for Music Generation

Monte-Carlo Tree Search (MCTS) is a heuristic search algorithm traditionally used to play board games with large search spaces. Before taking an action at the current game state, MCTS iteratively explores the branches of the search tree, looking ahead to determine how different game moves could lead to stronger or weaker states of the game. MCTS variations use a variety of algorithms for deciding which branches of the tree to explore next. For example, UCT [79] uses the UCB1 formula for deciding which branches to expand in each iteration of the algorithm, while AlphaZero uses the PUCT formula [125]. This section describes how this work employs PUCT to generate music with a specific target emotion.

PUCT receives a sequence $x = \{x_1, x_2, \cdots, x_{t-1}\}$ of musical tokens as input, which will bias the generative process; a music emotion classifier E; a language model L; a parameter k defining the number of top tokens (according to L) to consider when deciding the next token; a parameter d defining the number of search iterations PUCT can perform before deciding the next token; and a target emotion e. PUCT returns a sequence x' with prefix x for which $E(x', e)$ and $L(x')$ are maximized. PUCT grows a search tree where each node n represents a sequence of tokens from the vocabulary. The root of the tree represents the prefix x provided as input. Each edge (n, m) from node n to node m represents the addition of a token to the sequence n (m is one token larger than n). In this formulation, m is called a child of n, and since each node n represents a sequence x, n and x are used interchangeably. Initially, PUCT's tree is of size one as

it only contains the root of the tree. In each iteration, PUCT performs the following steps to add a new node to the tree: (1) **selection**, (2) **expansion**, (3) **simulation**, and (4) **backpropagation**.

1. **Selection**

   Starting from the current node n, PUCT recursively selects the token l that maximizes Equation 7.1 until l leads to a node m that is not in the PUCT tree. In Equation 7.1, $Q(n,l)$ is the expected emotion "reward" of the sequence (n,l) as given by E, c is an exploration constant, $P(n,l)$ is the prior probability of l being the next token from n as given by L, $N(n)$ is the number of times node n has been visited in the selection step, and $N(n,l)$ is the number of times token l has been chosen from node n in a selection step.

$$\arg\max_{l} Q(n,l) + c \times P(n,l) \times \frac{\sqrt{N(n)}}{1+N(n,l)} \tag{7.1}$$

   In practice, only the k tokens with the largest probability according to L are considered in the selection step. This allows PUCT to focus on the sequences that are more promising according to the LM.

2. **Expansion**

   The node m returned in the selection step is then added to the PUCT tree and its statistics are initialized: $N(m) = 1$, $N(m,l) = 0$, and $Q(m,l) = 0$ for all top k tokens l according to the probability $L(m,l)$.

3. **Simulation**

The recently added node m is evaluated according to the target emotion e. The $Q(n,l)$-value (recall that adding l to n generated the node m) is given by $E(m,e)$. The value of $N(n,l)$ is set to 1 as this is the first time l is selected from node n.

4. **Backpropagation**

   The value of $Q(n,l)$ is used to update the Q-values of all the other node-token pairs recursively selected in the **selection step**. This is achieved by following the path in the tree from n to m in reverse order and updating the statistics of each node n in the path according to Equation 7.2.

$$\begin{aligned} Q(n,l) &= \frac{Q(n,l) \times N(n,l) + E(m,e)}{N(n,l)+1} \\ N(n,l) &= N(n,l)+1 \\ N(n) &= N(n)+1 \end{aligned} \tag{7.2}$$

   The $Q(n,l)$-values are the average E-values of the sequences with prefix given by n. In other words, the value of $Q(n,l)$ is the average emotion "reward" of the sequences with the prefix given by n. The backpropagation step completes an iteration of PUCT.

In the next iteration, PUCT will perform the four steps described above, but with updated values of N and Q for the node-token pairs selected in the previous iteration. Equation 7.1 guarantees that the sequences n that maximize the value of $E(n,e)$ are visited more often as they will have larger values of Q. The PUCT formula also accounts for the probability given by the language model, giving preference to sequences with higher probability according to L. Finally, the term $\frac{\sqrt{N(n)}}{1+N(n,l)}$ certifies that all nodes have a chance of being explored by the search.

PUCT performs d iterations before deciding which token will be added to the sequence represented at the root of the tree. That is, the search budget of d iterations is to decide the next token of the sequence. Let n be the root of the tree. The token l that will be added to the sequence n is sampled from the distribution given by the values $\frac{N(n)}{\sum_l N(n,l)}$. The node m resulting from the addition of l to n becomes the new root of the tree and another PUCT search is performed with budget d to choose the next token to be added to m. This process is repeated until a desired number of tokens are generated. The PUCT search can be seen as an operator that changes the probability distribution over tokens given by the language model such that it accounts for the target emotion. This is because the distribution given by $\frac{N(n)}{\sum_l N(n,l)}$ will favor tokens that lead to pieces matching the target emotion as nodes representing such pieces are visited more often during search.

7.5 Empirical Evaluation

MCTS is evaluated with two listening tests, one for measuring the quality of the generated pieces and one for measuring the accuracy of MCTS in conveying a target emotion. All experiments are performed via Amazon Mechanical Turk (MTurk). For both experiments, MCTS is compared against SBBS, TopK sampling, and human-composed pieces. Although TopK sampling does not consider emotion, it is a good baseline for music quality. MCTS is not compared against the approach from Chapter 5 because that approach is limited to sentiment. To generate the pieces to be evaluated, 10 prime sequences are selected from the VGMIDI dataset for each of the 4 emotions. Each prime sequence is then used to generate a piece with

each of the 4 models, yielding $10 \times 4 \times 4 = 160$ pieces. Each piece has 512 tokens. Each prime sequence is 32 tokens long and is selected at random from the VGMIDI test set with the target emotion e. For the human method, pieces are "generated" by simply extracting the first 512 tokens of the piece with the given prime.

To generate the 160 pieces, a Music Transformer LM is first trained with 4 layers (transformer blocks), a maximum sequence length of 2048 tokens, 8 attention heads, and an embedding layer with 384 units. The size of the Feed-Forward layers in each transformer block is set to 1024. This music transformer LM is trained with the 3,640 unlabelled pieces of the extended VGMIDI dataset, where 3,094 (85%) of the pieces are used for training and 546 (15%) for testing. All unlabelled pieces are augmented by (a) transposing to every key, (b) increasing and decreasing the tempo by 10%, and (c) increasing and decreasing the velocity of all notes by 10% [105]. The emotion classifier is then trained by fine-tuning the music transformer LM with an extra linear classification layer on top. The emotion classifier is trained with the 200 labeled pieces of the VGMIDI dataset, where 140 (70%) pieces are used for training and 30 (30%) for testing. After training, the losses of the music transformer LM are 0.54 on the training set and 0.73 on the test set. The accuracy of the emotion classifier on the test set is 61%. At generation time, the LM distribution is filtered with $k = 128$ in MCTS, SBBS, and TopK. The beam size for SBBS is set to $b = 4$. For MCTS, the number of simulation steps is set to $d = 30$ and the exploration constant to $c = 16$.

7.5.1 Quality Listening Test

The quality listening test consists of a pairwise comparison that follows the methodology proposed by Huang et al. [63]. Human subjects were presented with two generated pieces from two different models that were given the same priming sequence. The two pieces were presented side-by-side, and the participants were asked to select which one is more musical using a 5-point Likert scale. In this scale, 1 means "Left piece is much more musical", 2 means "Left piece is slightly more musical", 3 means "Tie", 4 means "Right piece is slightly more musical" and 5 means "Right piece is much more musical". The order of the two pieces was randomized to avoid ordering bias. Each of the 240 pairs of generated pieces were evaluated by 3 MTurk workers. In order to reduce noise in the results (mainly caused by random choices in Amazon MTurk), a test evaluation is included for each human subject. This test is another pair of pieces to be evaluated with the same Likert scale, but one piece is a human-composed piece and the other one is sampled from the LM without TopK filtering and temperature equal to 1.5 (forcing the sample to have poor quality). The subjects are also asked to briefly justify their choice with 1-3 short sentences. Participants who failed the test evaluation (choosing the sampled piece as more musical) or didn't write explanations longer than 5 words were filtered out. In total, this experiment yielded 389 comparisons. Each pair was evaluated at least once.

Table 7.1 shows the results of the quality test. The top part of the table shows the number of wins, ties, and losses of one model against another. MCTS performed exactly like TopK sampling and outperformed SBBS by ten wins. Surprisingly, MCTS won against human-composed pieces 12 times and tied 9 times. SBBS performed worse than TopK sampling,

winning 26 times and losing 31. As expected, all models performed worse than human compositions. The bottom part of the table shows the percentage of wins, ties, and losses for one model against all other models. Percentages are reported because, due to the filtering of the participants, the amount of comparisons for each model is not the same. The aggregated results also show that MCTS performs better than SBBS and the same as TopK sampling. A Kruskal-Wallis H test of the subject choices (values from 1 to 5) shows a statistically significant difference between the models with $p = 1.5e-4 < 0.01$.

One-Vs-One		**Wins**	**Ties**	**Losses**
MCTS	**TopK**	25	13	25
MCTS	**SBBS**	34	8	24
MCTS	**Human**	12	9	41
TopK	**SBBS**	31	6	26
TopK	**Human**	15	7	45
SBBS	**Human**	15	8	45
One-Vs-Rest		**Wins %**	**Ties %**	**Losses %**
MCTS		38	15	47
SBBS		33	12	55
TopK		37	13	50
Human		66	12	22

Table 7.1: Results of the quality listening test. The top part of the table reports the number of wins, ties, and losses for a model against each other model. The results are stated with respect to the left model. For example, MCTS won against SBBS 34 times and lost to SBBS 24 times. The bottom part of the table shows the percentage of wins, ties, and losses for a model against all the others.

7.5.2 Emotion Listening Test

In the emotion listening test, human subjects were asked to annotate the generated pieces according to the circumplex model of emotion using the same tool designed to annotate

the VGMIDI dataset (see Section 5.2.1). An annotation result is a time series of valence-arousal pairs where each element corresponds to a chunk (a bar if the piece has 4/4 time signature) of the piece. For this experiment, 3 MTurk workers were assigned for each piece generated by the MCTS, SBBS, and TopK methods (total of 360 annotations). Human pieces are not reannotated because they are the ground truth data used to train the music emotion classifier that is base for both MCTS and SBBS. No annotations were filtered out in this experiment.

The accuracy of a method in conveying a target emotion is measured with the percentage of chunks in the annotations that match the target emotion. Each valence-arousal pair is mapped to an emotion label by getting the quadrant of that valence-arousal pair (see Section 7.3 for details). Table 7.2 reports the accuracy of each model for each emotion. Overall, MCTS outperformed TopK (no emotion control) by an average of 15%. MCTS performed similarly to SBBS, with slightly better average accuracy. It is important to highlight that MCTS was able to considerably improve the accuracy in conveying the two least represented emotions (E1 and E2) in the VGMIDI dataset. This is a great result since labelling symbolic music according to emotion is an expensive task.

Model	**E0**	**E1**	**E2**	**E3**	**Avg.**
MCTS	**72**	**52**	**37**	57	**54**
SBBS	67	41	30	**70**	52
TopK	61	18	26	53	39

Table 7.2: Accuracy of each model in conveying the target emotions E0, E1, E2 and E3.

TopK performed reasonably well on average (39%), but this was primarily due to its ability to convey the two most represented emotions in the training data (E0 and E3). These

two emotions are very likely more represented in the unlabelled data as well, which was used to train the LM. Even though TopK sampling does not control emotion, the prime sequences they used to generate the pieces had the target emotions. Therefore, TopK (like all other models) is conditioned with this prime sequence towards the desired emotion. However, because TopK does not consider emotion when generating pieces, eventually it starts sampling tokens that deviate from the target emotion.

The three models performed poorly on conveying E2, which encompasses basic emotions such as sad, depressed, and tired. These poor performances can be explained by the fact that class E2 has the least number of examples in the VGMIDI dataset. Moreover, as showed in Chapter 5, negative pieces can be considered ambiguous by human annotators. One could mitigate this problem by increasing the number of pieces of class E2 in the VGMIDI dataset.

Although MCTS performed similarly to SBBS in terms of emotion, MCTS outperformed SBBS considerably with respect to music quality. SBBS tends to generate repetitive pieces more often than MCTS once repetition maximizes both the probabilities of the LM and the music emotion classifier. Since SBBS does not do backtracking, when it generates a good pattern that is likely according to the LM and that conveys the target emotion, it tends to repeat that pattern. Figure 7.2 illustrates this problem with examples of pieces generated by MCTS and SBBS. These two pieces were given the same prime sequence [2] x with the target emotion E2 (low valence and arousal). MCTS developed x by creating 4 parts: an introduction from $t = 0$ to $t = 15$, a *Part A* from $t = 15$ to $t = 30$, a *Part B* from $t = 30$ to $t = 43$, and a *Part C* from $t = 43$ until the end. Parts A, B, and C are similar to each other but present different variations

[2]Note that the first 8 seconds of the two pieces are the same.

of the bass line presented in the last 3 seconds of the introduction. SBBS, on the other hand, simply repeated the bass line and the chord presented in x until the end of the piece. This is a case where SBBS repeated a given pattern that maximized the LM and the music emotion classifier probabilities. This example also shows how backtracking allows MCTS to look for different patterns in the search space.

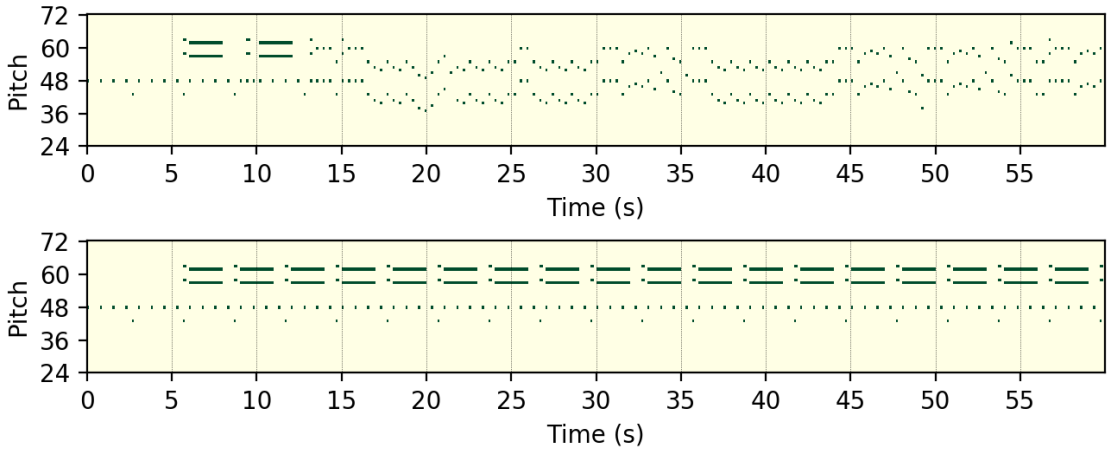


Figure 7.2: Examples of MCTS (top) and SBBS (bottom) pieces controlled to have emotion E2.

7.6 Expressivity Range

An expressivity analysis is performed to better understand how MCTS conveys different target emotions. Expressivity analysis is an evaluation method commonly used in the AI and Games research community to compare different level generators [127]. It consists of generating a large set of levels and measuring the biases of the generators according to pre-defined relevant metrics. In the music generation domain, this approach is used to explore the biases of the MCTS generator when conveying each emotion with respect to human compositions.

The expressivity analysis is performed on the 40 pieces generated by MCTS for the previous experiments, considering the frequencies of pitch classes and note durations as the metrics to measure bias. Figure 7.3 illustrates the distribution of pitch classes and note durations for both MCTS and human-composed pieces. Overall, the MCTS and human distributions of note durations look similar. For label E0 (high valence and arousal), human pieces predominantly have 16th notes, but a few 8th notes are also present. Quarter notes and 32nd notes are rarely used. MCTS also explored mainly 16th notes for label E0, however it used considerably more 8th and 32nd notes than the human compositions. Label E0 encapsulates emotions such as happy, delighted, and excited. These results show that both VGMIDI pieces and MCTS generated pieces with these emotions have rhythm patterns with shorter notes.

For label E1 (low valence and high arousal), human compositions have a more even distribution between 16th and 8th notes. Quarter notes are present but with a relatively low frequency, and 32nd notes are rarely used. MCTS also used a combination of 16th and 8th notes, but it used fewer quarter notes and a little more 32nd notes than the human compositions. Label E1 represents emotions such as tense, angry, and frustrated. Combining these results with the expressivity range of label E0, one concludes that the VGMIDI and MCTS pieces with high valence have rhythm patterns with notes shorter than a quarter note.

For label E2 (low valence and arousal), human pieces are mainly composed of quarter notes, with a few 8th and 16th notes. Different than E0 and E1, E2 has a few long notes such as half, whole, and breve notes. MCTS also focused on quarter notes for label E2. However, MCTS used more short notes and fewer long notes than human compositions. Label E2 encapsulates emotions such as sad, depressed, and tired. According to these results, both human and

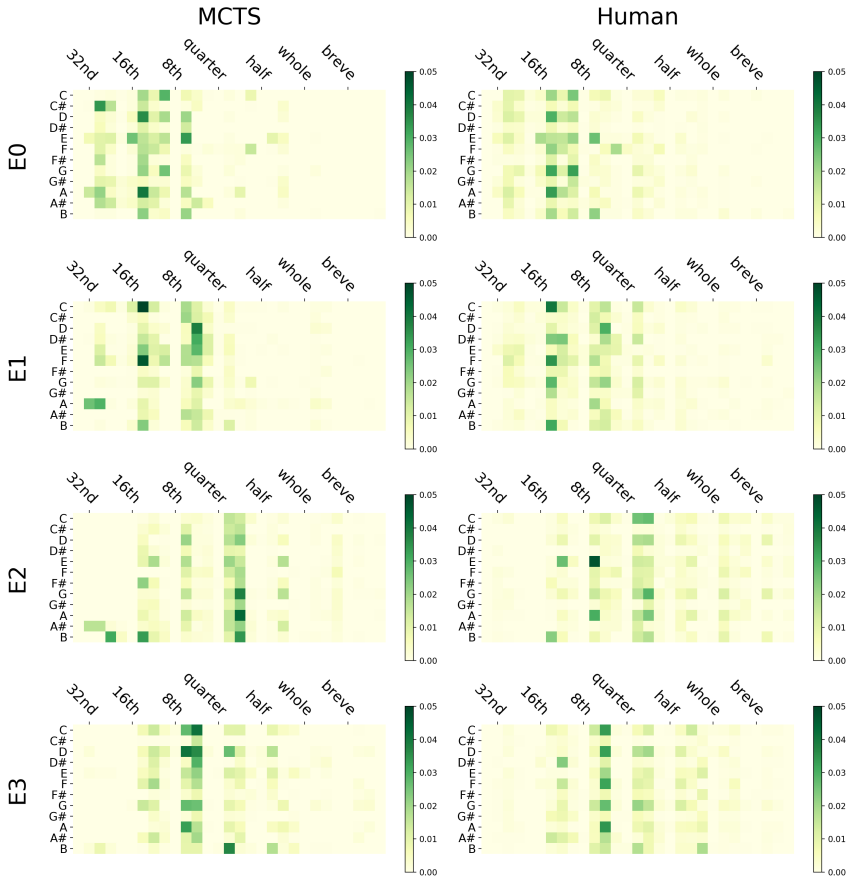


Figure 7.3: MCTS expressivity range. The x axis represents note duration in seconds and the y axis represents pitch classes.

MCTS pieces with these emotions tend to have rhythm patterns with long notes.

For label E3 (high valence and low arousal), both human and MCTS pieces have mainly 8th notes with a few 16h, quarter, and whole notes. These pieces represent emotions such as calm, relaxed, and content. With these expressivity results, one concludes that pieces with low arousal have rhythm patterns with long notes.

7.7 Conclusions

This chapter presented a decoding algorithm based on MCTS to generate symbolic music with controllable emotion. MCTS used PUCT to explore the search tree, where the probability of the nodes comes from a LM, and their scores are given by a music emotion classifier. MCTS was evaluated with two listening tests, one to measure the quality of the generated pieces and one to measure the accuracy of MCTS in conveying target emotions. In the first experiment, a pairwise comparison is performed between pieces generated by MCTS, SBBS, TopK sampling, and human composers. Human subjects were asked to rate which pieces they found more musical. In the second experiment, human subjects annotated the emotions they perceived in the generated pieces. Results showed that MCTS outperforms SBBS in terms of quality and has slightly better accuracy when controlling emotions. With an expressivity analysis, it was shown that MCTS generates pieces with music features similar to human compositions.

This is the first work to apply MCTS to decode neural LMs to generate symbolic music with controllable emotion. Given that MCTS is agnostic to the classifier used to steer the distribution of the LM, it can be used to control different features as long as a classifier can discriminate those features. Since MCTS generates music by searching over a large vocabulary with approximately 45K tokens, its generative task is similar to other NLP generative tasks. Therefore, MCTS can also be generalized to decode and control LMs in text generation tasks.

Chapter 8

Future Work

As discussed throughout this dissertation, the problem of composing music with a target emotion has been explored with different approaches, but only recently, with the rise of deep learning, NNs started to draw the attention of the AAC community. Indeed, this dissertation is one of the first works to approach AAC as a deep learning problem. Together with a few other recent works [90, 133, 152, 132], the contributions of this dissertation have opened up different possibilities of research that can be explored in the future. The remainder of this chapter describes different possibilities of future work that the AAC community can explore from this dissertation.

8.1 Datasets

Successful supervised deep learning systems typically are trained on hundreds of millions of labeled examples [84]. The available datasets for AAC, including the VGMIDI dataset, have only hundreds of labeled examples. One of the major contributions one could bring to

support deep NNs for AAC would be to extend the VGMIDI dataset or create a new dataset with a considerably larger number of labeled examples. Labeling symbolic music according to emotion is a subjective task because people can perceive emotions differently in music (as evidenced by the VGMIDI annotation process described in Chapter 5). This subjectivity makes the labeling task expensive once many annotators have to be assigned to the same piece in order to compute a meaningful (and democratic) final label. Therefore, another future work is to design a service or a game [82] where users would have an external incentive (other than money) to collaborate in the task of labeling music with a target emotion.

Most of the AAC datasets, including the VGMIDI, are limited to piano music. An important future work is to create datasets of multi-instrumental polyphonic music labeled according to emotion. These datasets would allow NNs to model the relationship between different timbres and perceived emotions. A significant challenge in creating such datasets is to collect a large collection of pieces that use the same set of instruments. Typically, the symbolic music datasets are created by gathering MIDI files publicly available on the Internet. Since these MIDI files come from different sources on the Internet, it is hard to guarantee consistency among the files. Once again, this problem could be addressed with an online service (or game), but instead of asking users to label pre-authored pieces, the system should support music composition with a given set of instruments and target emotions.

A specific problem with the VGMIDI dataset is that human subjects consider some of its negative pieces ambiguous. Thus, another future work consists of creating a dataset of less ambiguous pieces. This problem can be addressed by creating a dataset of movie soundtracks and asking annotators to label the pieces together with their respective scenes. The problem with

building soundtrack datasets is that most of these pieces are not publicly available in symbolic format. One way to solve this problem is to ask professional composers to create scores for a set of video clips that evoke different emotions.

8.2 Music Representation

Most datasets of symbolic music are organized as MIDI files. To learn models from these datasets, one has to create a vocabulary of music tokens extracted from the MIDI events or from the piano roll representation of the MIDI files. This dissertation explored vocabularies of different sizes. However, it is still unclear how different vocabularies affect the quality and controllability of the models. This analysis is a future work that should be performed with different decoding strategies (e.g., TopK, SBBS, and MCTS) and different data regimes (low data vs. big data). For example, one can investigate the impact of different vocabulary sizes on the level of repetition in pieces generated by different decoding algorithms. Another future work is to use a *vector quantized variational autoencoder* (VQ-VAE) to learn a vocabulary from MIDI files. Learning a vocabulary instead of manually defining one can yield better music representations that increase the overall quality and controllability of the models.

Although MIDI files provide easy access to high level music information (e.g., melody, harmony, and rhythm), they cannot capture human voices or many of the more subtle timbres, dynamics, and expressivity that the audio format can. The problem with modeling digital audio is that audio sequences are very long. For example, a typical four-minute song at CD quality (44 kHz, 16-bit) has over 10 million timesteps. The deep learning community has actively ex-

plored this problem [104, 93, 146], but the state-of-the-art methods still generate music that is significantly far from human composed music [27]. Designing NNs capable of modeling music from audio signals is an important future work that can potentially increase the expressivity of neural AAC systems.

8.3 Modeling

This dissertation explored three different approaches to control emotion in music generated by LMs. The first one is a genetic algorithm that optimizes the LM weights, and the other two are decoding strategies that steer the LM's distribution at generation time. These three approaches rely on pre-trained neural LMs with architectures especially designed to process sequences (e.g., RNNs and transformers). As future work, one can investigate how to control GANs and VAEs to compose music with a target emotion. GANs have shown great potential in music generation [100]. Moreover, conditional GANs have shown a great level of controllability in image generation [96]. One can train a conditional GAN to generate music where the conditions are input signals representing emotions in a categorical or dimensional model of emotion. The problem with using GANs for AAC is that training GANs with relatively small datasets typically leads to discriminator overfitting, causing training to diverge [72]. Regarding VAEs, one can extend Music FaderNets [132] to learn a representation between low-level music features and different emotions.

Another interesting direction of future work consists of fine-tuning generative models (e.g., transformers) with reinforcement learning. This can be done by pre-training a high-

capacity LM on a large unlabelled dataset of symbolic music and then using reinforcement learning to steer this LM towards different emotions. Ideally, the reinforcement learning fine-tuning should yield different policies for different emotions. Thus, after fine-tuning, the same LM could be used to compose music with different target emotions.

8.4 Decoding

SBBS and MCTS have shown strong potential to control emotion in music generated by LMs. As future work, one can apply these methods to generate sequences in different domains. For example, one can use MCTS to generate text, sketches, or video game levels. Moreover, one can investigate how to improve the quality of the music generated by both these methods – for example, exploring how to reduce repetition in the music pieces generated by SBBS. This can be done by training a repetition discriminator to model a common level of repetition in human composed music [59]. This discriminator should be combined with the emotion discriminator at decoding time. One can also apply trainable weights to each of these discriminators to linearly adjust their contributions to the final generated piece.

Section 4.2 showed different strategies to decode neural LMs in the natural language domain. Exploring these strategies to control emotion in generated music is another important direction of future work. For example, one can explore how to use the Plug and Play LM [23] to compose music with controllable emotion.

8.5 Applications

Bardo Composer is an important contribution towards generating affective music in real time for tabletop role-playing games. However, Bardo Composer was evaluated on the task of scoring videos of a D&D campaign. Thus, a future work consists of building a system to score D&D campaigns in real time and evaluating how the system affects the players' experiences. Another problem with Bardo Composer is that it uses two independent datasets of story and music, where each dataset has a different model of emotion [106]. Mapping emotions from one model to the other yields a hybrid model that is not as meaningful as the original ones. This problem can be handled in a future work where one creates an integrated dataset of music and stories. This dataset should use music pieces that are typically used with tabletop games. Moreover, it should be annotated according to a model of emotion that is meaningful for music and D&D stories.

One can also build upon Bardo Composer to generate soundtracks for other media. For example, one can directly apply the approach described in Chapter 6 to score audiobooks. One can also extract sentiment signals from video games scenes using game events (e.g., powerup pickup) and apply Bardo Composer to generate video game soundtracks or sound effects. Similarly, one could extract emotion signals from movies using raw pixels and apply Bardo Composer to generate movie soundtracks. Another exciting application of Bardo Composer is to include it as part of virtual assistants (e.g., Google Assistant or Siri) to allow users to have personalized soundtracks based on the emotional tone of their conversations.

Chapter 9

Conclusion

This dissertation presented three search-based approaches to control music language models to generate symbolic music with a target emotion. The first approach, described in Chapter 5, consists of a genetic algorithm to optimize an LSTM language model towards generating negative or positive pieces [42]. According to human evaluators, this approach showed to be successful in generating positive pieces, but negative pieces were considered slightly ambiguous. The second approach is called SBBS and is described in Chapter 6. It consists of controlling a stochastic beam search algorithm with emotion discriminators that steer the distribution of a language model towards a given emotion [41]. A listening test showed that human subjects could correctly identify the emotion of the pieces generated by SBBS as accurately as they were able to identify the emotion of pieces written by humans. The third approach, presented in Chapter 7, is a MCTS algorithm that uses PUCT with an emotion discriminator to search for music pieces with a target emotion. According to a listening test, MCTS outperformed SBBS in terms of quality and has slightly better accuracy when controlling emotions.

A dataset of symbolic music called VGMIDI has been created to support these three approaches. VGMIDI has 200 pieces labeled according to the circumplex model of emotion [121], and an additional 3,640 unlabelled pieces. All of them are piano arrangements of video game soundtracks. The labeling process was performed by 30 annotators with a custom web tool designed as part of this dissertation. Moreover, this dissertation also presented Bardo Composer, a system to generate music for tabletop role-playing games. Bardo Composer uses a speech recognition system to translate player speeches into captions, which Bardo Composer classifies according to a model of emotion. Bardo Composer then uses SBBS with a neural music emotion classifier to generate pieces conveying the emotion detected in the captions.

The contributions of this dissertation showed that searching over the space defined by music language models with the guidance of music emotion classifiers has strong potential in generating music with controllable emotion. As discussed in Chapter 8, there are several possibilities for future work, from increasing the VGMIDI dataset to developing new generative models and applying Bardo Composer to other media (e.g., video games and films). Hopefully, the contributions of this dissertation will inspire others to explore some of this future work and other ideas that will enable generative models to compose music with human level quality.

Appendix A

Reproducibility

List of references to reproduce the work in this dissertation:

Learning to Generate Music with Sentiment
VGMIDI Dataset: `https://github.com/lucasnfe/vgmidi`
VGMIDI Music Annotation: `https://github.com/lucasnfe/adl-music-annotation`
Source Code: `https://github.com/rafaelpadovani/music-sentneuron`

Computer Generated Music for Tabletop Role-Playing Games
VGMIDI Dataset: `https://github.com/lucasnfe/vgmidi`
ADL Piano MIDI Dataset: `https://github.com/lucasnfe/adl-piano-midi`
Call of the Wild Dataset: `https://github.com/lucasnfe/bardo`
Source Code: `https://github.com/lucasnfe/bardo-composer`

Controlling Emotions in Symbolic Music Generation with MCTS
VGMIDI Dataset: `https://github.com/lucasnfe/vgmidi`
Source Code: `https://github.com/lucasnfe/mucts`

Bibliography

[1] International e-piano competition. http://www.piano-e-competition.com. Accessed: 2019-04-12.

[2] Charles Ames. Automated composition in retrospect: 1956-1986. *Leonardo*, pages 169–185, 1987.

[3] Charles Ames. The markov process as a compositional model: A survey and tutorial. *Leonardo*, 22(2):175–187, 1989.

[4] Torsten Anders and Eduardo R Miranda. Constraint programming systems for modeling music theories and composition. *ACM Computing Surveys (CSUR)*, 43(4):1–38, 2011.

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] Philip Ball. Making music by numbers online, 2005.

[8] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

[9] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The journal of machine learning research*, 3:1137–1155, 2003.

[10] Yoshua Bengio, Aaron C Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR, abs/1206.5538*, 1:2012, 2012.

[11] Karl Bergström and Staffan Björk. The case for computer-augmented games. *Transactions of the Digital Games Research Association*, 1(3), 2014.

[12] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. The million song dataset. 2011.

[13] John Biles et al. Genjam: A genetic algorithm for generating jazz solos. In *ICMC*, volume 94, pages 131–137. Ann Arbor, MI, 1994.

[14] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

[15] Margaret M Bradley, Mark K Greenwald, Margaret C Petry, and Peter J Lang. Remembering pictures: pleasure and arousal in memory. *Journal of experimental psychology: Learning, Memory, and Cognition*, 18(2):379, 1992.

[16] Jean-Pierre Briot, Gaëtan Hadjeres, and François Pachet. Deep learning techniques for music generation-a survey. *arXiv preprint arXiv:1709.01620*, 2017.

[17] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[18] F. P. Brooks, A. L. Hopkins, P. G. Neumann, and W. V. Wright. An experiment in musical composition. *IRE Transactions on Electronic Computers*, EC-6(3):175–182, 1957. doi: 10.1109/TEC.1957.5222016.

[19] Gino Brunner, Andres Konrad, Yuyi Wang, and Roger Wattenhofer. Midi-vae: Modeling dynamics and instrumentation of music with applications to style transfer. *arXiv preprint arXiv:1809.07600*, 2018.

[20] Heather Chan and Dan Ventura. Automatic composition of themed mood pieces. In *Proceedings of the 5th International Joint Workshop on Computational Creativity*, pages 109–115. Citeseer, 2008.

[21] Sixian Chen, John Bowers, and Abigail Durrant. 'ambient walk': A mobile application for mindful walking with sonification of biophysical data. In *Proceedings of the 2015 British HCI Conference*, British HCI '15, pages 315–315, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3643-7. doi: 10.1145/2783446.2783630. URL `http://doi.acm.org/10.1145/2783446.2783630`.

[22] David Cope. Experiments in musical intelligence (emi): Non-linear linguistic-based composition. *Journal of New Music Research*, 18(1-2):117–139, 1989.

[23] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: a simple approach to controlled text generation. *arXiv preprint arXiv:1912.02164*, 2019.

[24] Hannah Davis and Saif M Mohammad. Generating music from literature. *Proceedings of the 3rd Workshop on Computational Linguistics for Literature (CLfL)*, pages 1–10, 2014.

[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[26] A. K. Dewdney. Computer recreations. *Scientific American*, 261(2):102–106, 1989. ISSN 00368733, 19467087. URL `http://www.jstor.org/stable/24987368`.

[27] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

[28] Chris Donahue, Huanru Henry Mao, and Julian McAuley. The nes music database: A multi-instrumental dataset with expressive performance attributes. In *ISMIR*, 2018.

[29] Chris Donahue, Huanru Henry Mao, Yiting Ethan Li, Garrison W Cottrell, and Julian McAuley. Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training. *arXiv preprint arXiv:1907.04868*, 2019.

[30] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[31] MO Duff. Backpropagation and bach's 5th cello suite (sarabande). In *International 1989 Joint Conference on Neural Networks*, pages 575–vol. IEEE, 1989.

[32] Kemal Ebcioğlu. An expert system for harmonizing four-part chorales. *Computer Music Journal*, 12(3):43–51, 1988.

[33] Tuomas Eerola and Jonna K Vuoskoski. A comparison of the discrete and dimensional models of emotion in music. *Psychology of Music*, 39(1):18–49, 2011.

[34] Panteleimon Ekkekakis. *The measurement of affect, mood, and emotion: A guide for health-behavioral research*. Cambridge University Press, 2013.

[35] Paul Ekman. An argument for basic emotions. *Cognition & emotion*, 6(3-4):169–200, 1992.

[36] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[37] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. Neural audio synthesis of musical notes with wavenet autoencoders. In *International Conference on Machine Learning*, pages 1068–1077. PMLR, 2017.

[38] B Eno. Generative music, speech at the imagination conference, 1996.

[39] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*, 2018.

[40] Andy Farnell. An introduction to procedural audio and its application in computer games. In *Audio mostly conference*, volume 23. Citeseer, 2007.

[41] Lucas Ferreira, Levi Lelis, and Jim Whitehead. Computer-generated music for tabletop role-playing games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 59–65, 2020.

[42] Lucas N Ferreira and Jim Whitehead. Learning to generate music with sentiment. In *Proceedings of the International Society for Music Information Retrieval Conference*, ISMIR'19, 2019. URL http://www.lucasnferreira.com/papers/2019/ismir-music-sentneuron.pdf.

[43] Anders Friberg, Roberto Bresin, and Johan Sundberg. Overview of the kth rule system for musical performance. *Advances in cognitive psychology*, 2(2):145, 2006.

[44] Alf Gabrielsson. Emotion perceived and emotion felt: Same or different? *Musicae scientiae*, 5(1_suppl):123–147, 2001.

[45] Martin Gardner. Mathematical games. *Scientific American*, 223(4):120–123, 1970. ISSN 00368733, 19467087. URL http://www.jstor.org/stable/24927642.

[46] Stanley Gill. A technique for the composition of music in a computer. *The Computer Journal*, 6(2):129–133, 1963.

[47] Jon Gillick, Kevin Tang, and Robert M Keller. Machine learning of jazz grammars. *Computer Music Journal*, 34(3):56–66, 2010.

[48] Michael Good. Musicxml for notation and analysis. *The virtual score: representation, retrieval, restoration*, 12(113-124):160, 2001.

[49] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[50] Ian J Goodfellow, Dumitru Erhan, Pierre Luc Carrier, Aaron Courville, Mehdi Mirza, Ben Hamner, Will Cukierski, Yichuan Tang, David Thaler, Dong-Hyun Lee, et al. Challenges in representation learning: A report on three machine learning contests. In *International conference on neural information processing*, pages 117–124. Springer, 2013.

[51] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[52] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1362–1371. JMLR. org, 2017.

[53] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=r1lYRjC9F7.

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[55] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 507–517, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4143-1. doi: 10.1145/2872427.2883037. URL https://doi.org/10.1145/2872427.2883037.

[56] Hermann Hild, Johannes Feulner, and Wolfram Menzel. Harmonet: A neural net for harmonizing chorales in the style of js bach. In *Advances in neural information processing systems*, pages 267–274, 1992.

[57] Lejaren A Hiller Jr and Leonard M Isaacson. Musical composition with a high speed digital computer. In *Audio Engineering Society Convention 9*. Audio Engineering Society, 1957.

[58] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[59] Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. Learning to write with cooperative discriminators. *arXiv preprint arXiv:1805.06087*, 2018.

[60] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[61] SR Holtzman. Using generative grammars for music composition. *Computer Music Journal*, 5(1):51–64, 1981.

[62] Andrew Horner and David E Goldberg. *Genetic algorithms and computer-assisted music composition*, volume 51. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 1991.

[63] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. *arXiv preprint arXiv:1809.04281*, 2018.

[64] Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Douglas Eck. Counterpoint by convolution. *arXiv preprint arXiv:1903.07227*, 2019.

[65] Yu-Siang Huang and Yi-Hsuan Yang. Pop music transformer: Generating music with rhythm and harmony. *arXiv preprint arXiv:2002.00212*, 2020.

[66] Bruce Jacob. Composing with genetic algorithms. 1995.

[67] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[68] Dasaem Jeong, Taegyun Kwon, Yoojin Kim, Kyogu Lee, and Juhan Nam. Virtuosonet: A hierarchical rnn-based system for modeling expressive piano performance. In *ISMIR*, pages 908–915, 2019.

[69] Daniel D Johnson. Generating polyphonic music using tied parallel networks. In *International conference on evolutionary and biologically inspired music and art*, pages 128–143. Springer, 2017.

[70] Kevin Jones. Compositional applications of stochastic processes. *Computer Music Journal*, 5(2):45–61, 1981.

[71] Michael I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546, 1986.

[72] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. *arXiv preprint arXiv:2006.06676*, 2020.

[73] Robert M Keller and David R Morrison. A grammatical approach to automatic improvisation. In *Proceedings, Fourth Sound and Music Conference, Lefkada, Greece, July.Most of the soloists at Birdland had to wait for Parkers next record in order to find out what to play next. What will they do now*. Citeseer, 2007.

[74] Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

[75] Sunjung Kim and Elisabeth André. Composing affective music with a generate and sense approach. In *FLAIRS Conference*, pages 38–43, 2004.

[76] Youngmoo E Kim, Erik M Schmidt, Raymond Migneco, Brandon G Morton, Patrick Richardson, Jeffrey Scott, Jacquelin A Speck, and Douglas Turnbull. Music emotion recognition: A state of the art review. In *Proc. ISMIR*, volume 86, pages 937–952. Citeseer, 2010.

[77] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[78] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[79] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the European Conference on Machine Learning*, pages 282–293, Berlin, Heidelberg, 2006. Springer-Verlag.

[80] Ben Krause, Iain Murray, Steve Renals, and Liang Lu. Multiplicative LSTM for sequence modelling. *ICLR Workshop track*, 2017.

[81] Peter Langston. Six techniques for algorithmic music composition. In *Proceedings of the International Computer Music Conference*, volume 60. Citeseer, 1989.

[82] Edith Law, L. V. Ahn, R. Dannenberg, and Mike Crawford. Tagatune: A game for music and sound annotation. In *ISMIR*, 2007.

[83] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[84] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

[85] Fred Lerdahl and Ray S Jackendoff. *A Generative Theory of Tonal Music, reissue, with a new preface*. MIT press, 1996.

[86] Feynman T Liang, Mark Gotham, Matthew Johnson, and Jamie Shotton. Automatic stylistic composition of bach chorales with deep lstm. In *ISMIR*, pages 449–456, 2017.

[87] David Lidov and Jim Gabura. A melody writing algorithm using a formal language model. *Computer Studies in the Humanities*, 4(3-4):138–148, 1973.

[88] ManYat Lo and Simon M Lucas. Evolving musical sequences with n-gram based trainable fitness functions. In *2006 ieee international conference on evolutionary computation*, pages 601–608. IEEE, 2006.

[89] Sergio Luque. The stochastic synthesis of iannis xenakis. *Leonardo Music Journal*, 19: 77–84, 2009.

[90] Rishi Madhok, Shivali Goel, and Shweta Garg. Sentimozart: Music generation based on emotions. In *ICAART (2)*, pages 501–506, 2018.

[91] Huanru Henry Mao, Taylor Shin, and Garrison Cottrell. Deepj: Style-specific music generation. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*, pages 377–382, 2018. doi: 10.1109/ICSC.2018.00077.

[92] Ryan A McIntyre. Bach in a box: The evolution of four part baroque harmony using the genetic algorithm. In *Proceedings of the first ieee conference on evolutionary computation. ieee world congress on computational intelligence*, pages 852–857. IEEE, 1994.

[93] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*, 2016.

[94] Eduardo R Miranda, Wendy L Magee, John J Wilson, Joel Eaton, and Ramaswamy Palaniappan. Brain-computer music interfacing (bcmi): from basic research to the real world of special needs. *Music & Medicine*, 3(3):134–140, 2011.

[95] Eduardo Reck Miranda. Cellular automata music: An interdisciplinary project. *Journal of New Music Research*, 22(1):3–21, 1993.

[96] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[97] Olof Mogren. C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*, 2016.

[98] Kristine Monteith, Tony R Martinez, and Dan Ventura. Automatic generation of music for inducing emotive response. In *International Conference on Computational Creativity*, pages 140–149, 2010.

[99] James Anderson Moorer. Music and computer composition. *Communications of the ACM*, 15(2):104–113, 1972.

[100] Aashiq Muhamed, Liang Li, Xingjian Shi, Suri Yaddanapudi, Wayne Chi, Dylan Jackson, Rahul Suresh, Zachary C Lipton, and Alexander J Smola. Symbolic music generation with transformer-gans. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 408–417, 2021.

[101] Gerhard Nierhaus. *Algorithmic composition: paradigms of automated music generation*. Springer Science & Business Media, 2009.

[102] Kota Nomura and Makoto Fukumoto. Music melodies suited to multiple users' feelings composed by asynchronous distributed interactive genetic algorithm. *International Journal of Software Innovation (IJSI)*, 6(2):26–36, 2018.

[103] Christopher Olah. Understanding lstm networks. 2015.

[104] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[105] Sageev Oore, Ian Simon, Sander Dieleman, and Doug Eck. Learning to create piano performances. In *NIPS 2017 Workshop on Machine Learning for Creativity and Design*, 2017.

[106] Rafael Padovani, Lucas N. Ferreira, and Levi H. S. Lelis. Bardo: Emotion-based music recommendation for tabletop role-playing games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2017.

[107] Renato Eduardo Silva Panda, Ricardo Malheiro, Bruno Rocha, António Pedro Oliveira, and Rui Pedro Paiva. Multi-modal music emotion recognition: A new dataset, methodology and comparative analysis. In *10th International Symposium on Computer Music Multidisciplinary Research (CMMR 2013)*, pages 570–582, 2013.

[108] George Papadopoulos, Geraint Wiggins, et al. A genetic algorithm for the generation of jazz melodies. *Proceedings of STEP*, 98, 1998.

[109] W Gerrod Parrott. *Emotions in social psychology: Essential readings*. psychology press, 2001.

[110] Francisco C Pereira, Carlos Fernando Almeida Grilo, Luís Macedo, and Fernando Amílcar Bandeira Cardoso. Composing music with case-based reasoning. In *International Conference on Computational Models of Creative Cognition*, 1997.

[111] John Polito, Jason M Daida, and Tommaso F Bersano-Begey. Musica ex machina: Composing 16th-century counterpoint with genetic programming and symbiosis. In *International Conference on Evolutionary Programming*, pages 113–123. Springer, 1997.

[112] David L Poole and Alan K Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.

[113] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.

[114] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. In *Arxiv*, 2018.

[115] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[116] Colin Raffel. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. PhD thesis, Columbia University, 2016.

[117] Adhika Sigit Ramanto and Nur Ulfa Maulidevi. Markov chain based procedural music generator with user chosen mood compatibility. *International Journal of Asia Digital Art and Design Association*, 21(1):19–24, 2017.

[118] Adam Roberts, Jesse Engel, and Douglas Eck, editors. *Hierarchical Variational Autoencoders for Music*, 2017. URL `https://nips2017creativity.github.io/doc/Hierarchical_Variational_Autoencoders_for_Music.pdf`.

[119] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. In *International Conference on Machine Learning*, pages 4364–4373. PMLR, 2018.

[120] Joseph Rocca. Understanding variational autoencoders (vaes). *Data Science*, 2019.

[121] James A Russell. A circumplex model of affect. *Journal of personality and social psychology*, 39(6):1161, 1980.

[122] Harald Schmidl. Pseudo-genetic algorithmic composition. In *GEM*, pages 24–29. Citeseer, 2008.

[123] Marco Scirea, Julian Togelius, Peter Eklund, and Sebastian Risi. Affective evolutionary music composition with metacompose. *Genetic Programming and Evolvable Machines*, 18(4):433–465, 2017.

[124] Thalles Silva. An intuitive introduction to generative adversarial networks (gans). *freeCodeCamp. org*, 2018.

[125] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[126] Mary Simoni. *Algorithmic composition: a gentle introduction to music composition using common LISP and common music*. Michigan Publishing, University of Michigan Library, 2003.

[127] Gillian Smith and Jim Whitehead. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, pages 1–7, 2010.

[128] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[129] Mohammad Soleymani, Micheal N. Caro, Erik M. Schmidt, Cheng-Ya Sha, and Yi-Hsuan Yang. 1000 songs for emotional analysis of music. In *Proceedings of the 2Nd ACM International Workshop on Crowdsourcing for Multimedia*, CrowdMM '13, pages 1–6, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2396-3. doi: 10.1145/2506364.2506365. URL `http://doi.acm.org/10.1145/2506364.2506365`.

[130] Makis Solomos. Cellular automata in xenakis's music. theory and practice. In *International Symposium Iannis Xenakis (Athens, May 2005)*, pages 11–p, 2005.

[131] Simeng Sun and Mohit Iyyer. Revisiting simple neural probabilistic language models. *arXiv preprint arXiv:2104.03474*, 2021.

[132] Hao Hao Tan and Dorien Herremans. Music fadernets: Controllable music generation based on high-level features via low-level feature modelling. *arXiv preprint arXiv:2007.15474*, 2020.

[133] Xiaodong Tan and Mathis Antony. Automated music generation for visual art through emotion. In *ICCC*, pages 247–250, 2020.

[134] Sever Tipei. *MP1: a computer program for music composition*. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 1975.

[135] Peter M Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43, 1989.

[136] Nao Tokui, Hitoshi Iba, et al. Music composition with interactive evolutionary computation. In *Proceedings of the third international conference on generative art*, volume 17, pages 215–226, 2000.

[137] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[138] Patrick von Platen. How to generate text: using different decoding methods for language generation with transformers. `https://huggingface.co/blog/how-to-generate`, 2020. [Online; accessed 05-July-2021].

[139] Chris Walshaw. Abc2mtex: An easy way of transcribing folk and traditional music, version 1.0. *University of Greenwich, London*, 1993.

[140] Duncan Williams, Alexis Kirke, Joel Eaton, Eduardo Miranda, Ian Daly, James Hallowell, Etienne Roesch, Faustina Hwang, and Slawomir J Nasuto. Dynamic game soundtrack generation in response to a continuously varying emotional trajectory. In *Audio Engineering Society Conference: 56th International Conference: Audio for Games*. Audio Engineering Society, 2015.

[141] Duncan Williams, Alexis Kirke, Eduardo R Miranda, Etienne Roesch, Ian Daly, and Slawomir Nasuto. Investigating affect in algorithmic composition systems. *Psychology of Music*, 43(6):831–854, 2015.

[142] Duncan AH Williams, Victoria J Hodge, Chia-Yu Wu, et al. On the use of ai for generation of functional music to improve mental health. *Frontiers in Artificial Intelligence*, 2020.

[143] Stephen Wolfram. *A new kind of science*, volume 5. Wolfram media Champaign, IL, 2002.

[144] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[145] Iannis Xenakis. *Formalized music: thought and mathematics in composition*. Number 6. Pendragon Press, 1992.

[146] Ryuichi Yamamoto, Eunwoo Song, and Jae-Min Kim. Parallel wavegan: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6199–6203. IEEE, 2020.

[147] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv:1703.10847*, 2017.

[148] Ruihan Yang, Dingsu Wang, Ziyu Wang, Tianyao Chen, Junyan Jiang, and Gus Xia. Deep music analogy via latent representation disentanglement. *arXiv preprint arXiv:1906.03626*, 2019.

[149] Quanzeng You, Jiebo Luo, Hailin Jin, and Jianchao Yang. Building a large scale dataset for image emotion recognition: The fine print and the benchmark. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[150] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[151] Marcel Zentner, Didier Grandjean, and Klaus R Scherer. Emotions evoked by the sound of music: characterization, classification, and measurement. *Emotion*, 8(4):494, 2008.

[152] Kun Zhao, Siqi Li, Juanjuan Cai, Hui Wang, and Jingling Wang. An emotional symbolic music generation system based on lstm networks. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 2039–2043. IEEE, 2019.

[153] Hua Zhu, Shangfei Wang, and Zhen Wang. Emotional music generation using interactive genetic algorithm. In *2008 International Conference on Computer Science and Software Engineering*, volume 1, pages 345–348. IEEE, 2008.

[154] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.