

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

Copyright © by

Lucas N. Ferreira

2021

# Table of Contents

<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>ix</b>
<b>Abstract</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Neural Language Models . . . . .	2
1.2 Affective Algorithmic Composition . . . . .	4
1.3 Contributions . . . . .	5
1.3.1 Learning to Generate Music with Sentiment . . . . .	5
1.3.2 Computer-Generated Music for Tabletop Role-Playing Games . . . . .	6
1.3.3 Controlling Emotions in Symbolic Music Generation with MCTS . . . . .	7
1.4 Dissertation Outline . . . . .	8
<b>2 A Brief History of Algorithmic Music Composition</b>	<b>10</b>
2.1 Procedures in Music Theory . . . . .	11
2.2 Computational Methods . . . . .	18
2.2.1 Expert Systems . . . . .	19
2.2.2 Generative Grammars . . . . .	20
2.2.3 Cellular Automata . . . . .	21
2.2.4 Evolutionary Algorithms . . . . .	21
2.2.5 Markov Chains . . . . .	22
<b>3 Deep Learning for Algorithmic Music Composition</b>	<b>26</b>
3.1 Symbolic Music Representation . . . . .	28
3.1.1 MIDI . . . . .	28
3.1.2 Piano Roll . . . . .	29
3.2 Neural Networks . . . . .	31
3.2.1 Recurrent Neural Networks . . . . .	36
3.2.2 Long Short-Term Memory Networks . . . . .	38
3.2.3 Transformers . . . . .	41

3.2.4	Variational Autoencoders . . . . .	46
3.2.5	Generative Adversarial Networks . . . . .	50
3.3	Decoding . . . . .	53
3.3.1	Top-k Sampling . . . . .	54
3.3.2	Top-p (Nucleus) Sampling . . . . .	56
3.3.3	Greedy Search . . . . .	57
3.3.4	Beam Search . . . . .	58
<b>4</b>	<b>Controllable Algorithmic Music Composition</b>	<b>61</b>
4.1	Affective Algorithmic Composition . . . . .	61
4.1.1	Models of Emotion . . . . .	62
4.1.2	Expert systems . . . . .	65
4.1.3	Evolutionary Algorithms . . . . .	66
4.1.4	Markov Chains . . . . .	68
4.1.5	Deep Learning . . . . .	69
4.2	Controllable Neural Language Models . . . . .	72
<b>5</b>	<b>Learning to Generate Music with Sentiment</b>	<b>75</b>
5.1	Introduction . . . . .	75
5.2	The VGMIDI dataset . . . . .	76
5.2.1	Annotation Tool and Data Collection . . . . .	77
5.2.2	Data Analysis . . . . .	78
5.3	Models and Data Representation . . . . .	80
5.4	Sentiment Analysis Evaluation . . . . .	82
5.5	Generative Evaluation . . . . .	86
5.6	Conclusions . . . . .	89
<b>6</b>	<b>Computer-Generated Music for Tabletop Role-Playing Games</b>	<b>91</b>
6.1	Introduction . . . . .	91
6.2	Bardo Composer . . . . .	94
6.2.1	Classifying the Story’s Emotion . . . . .	95
6.2.2	Classifying the Music’s Emotion . . . . .	96
6.2.3	Stochastic Bi-Objective Beam Search . . . . .	98
6.3	Empirical Evaluation . . . . .	101
6.3.1	Emotion Classifiers . . . . .	101
6.3.2	User Study . . . . .	105
6.4	Conclusions . . . . .	108
<b>7</b>	<b>Controlling Emotions in Symbolic Music Generation with MCTS</b>	<b>109</b>
7.1	Introduction . . . . .	109
7.2	Language Model . . . . .	111
7.3	Emotion Classification Model . . . . .	112
7.4	MCTS for Music Generation . . . . .	114
7.5	Experiments . . . . .	117

7.5.1	Quality . . . . .	118
7.5.2	Emotion . . . . .	119
7.6	Expressivity Range . . . . .	123
7.7	Conclusions . . . . .	125
<b>8</b>	<b>Future Work</b>	<b>127</b>
8.1	Datasets . . . . .	127
8.2	Music Representation . . . . .	129
8.3	Modeling . . . . .	130
8.4	Decoding . . . . .	131
8.5	Applications . . . . .	132
<b>9</b>	<b>Conclusion</b>	<b>134</b>

# List of Figures

2.1	Mapping of vowels on tone pitches by Guido of Arezzo. . . . .	12
2.2	The tenor of <i>De bon espoir-Puisque la douce-Speravi</i> by Guillaume de Machaut. . . . .	12
2.3	<i>Canon per Augmentationem in Contrario Motu</i> from <i>The Art of the Fugue</i> by Johann Sebastian Bach. . . . .	13
2.4	Subject of the canon. . . . .	14
2.5	Example of minuet generated using Mozart’s dice game. . . . .	15
2.6	Tone row used by Alban Berg in the <i>Lyric Suite</i> for string quartet composed in 1926. . . . .	16
2.7	Example of Markov Chain represented as a directed. . . . .	23
3.1	Example of music represented in a piano roll format. . . . .	30
3.2	Diagram of a feedforward neural network [81]. . . . .	33
3.3	Diagram of a RNN [98]. . . . .	36
3.4	Diagram of a LSTM [98]. . . . .	39
3.5	Diagram of a Transformer [133]. . . . .	42
3.6	Diagram of an Autoencoder [116]. . . . .	47

3.7	Sampling from the latent space built by an Autoencoder [116]. . . . .	47
3.8	Diagram of a Variational Autoencoder (VAE) [116]. . . . .	48
3.9	Diagram of a Generative Adversarial Network (GAN) [121]. . . . .	50
3.10	Example of sampling for text generation with prior $x = [The]$ [134]. . . . .	54
3.11	Example of sampling for text generation with temperature $t = 0.7$ and prior $x = [The]$ [134]. . . . .	55
3.12	Example top-k sampling for text generation with $k = 6$ and prior $x = [The]$ [134].	55
3.13	Example top-p sampling for text generation with prior $x = [The]$ [134]. . . . .	57
3.14	Example of greedy search for text generation with prior $x = [The]$ [134]. . . . .	58
3.15	Example of beam search for text generation with prior $x = [The]$ and beam size $b = 2$ [134]. . . . .	59
4.1	The circumplex model of emotion. The horizontal and vertical axes represent valence and arousal, respectively [118]. . . . .	64
5.1	Screenshot of the annotation tool. . . . .	79
5.2	Data analysis process used to define the final label of the phrases of a piece. . .	81
5.3	A short example piece encoded using the proposed representation. The encod- ing represents the first two time steps of the shown measure. . . . .	83
5.4	Weights of 161 L1 neurons. Note multiple prominent positive and negative neurons. . . . .	87
5.5	Average valence of the 6 generated pieces, as determined by human annotators. with least variance. . . . .	89

6.1	Diagrams with the architecture of Bardo (left) and Composer (right). . . . .	93
7.1	Mapping the circumplex model to a categorical model of emotion with four classes: E1, E2, E3, and E4. . . . .	113
7.2	Examples of MCTS (top) and SBBS (bottom) pieces controlled to have emotion E2. . . . .	122
7.3	MCTS expressivity range. The x axis represents note duration in seconds and the y axis represents pitch classes. . . . .	124



# List of Tables

2.1	Example of Mozart’s dice game. Each element in the table is an integer representing the number of a pre-composed measure. . . . .	15
2.2	Different transformations that can be applied to the tone row showed in Figure 2.6. . . . .	17
3.1	Example of MIDI file encoded in a readable format. . . . .	29
3.2	A list of common activation functions in the hidden layer. . . . .	34
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.” [133]. . . . .	44
5.1	Average cross entropy loss of the generative mLSTM with different amount of neurons. . . . .	85
5.2	Average (10-fold cross validation) sentiment classification accuracy of both generative (with logistic regression) and supervised mLSTMs. . . . .	85
6.1	Valence accuracy in % of Naïve Bayes (NB) and BERT for story emotion classification. . . . .	102

6.2	Arousal accuracy in % of Naïve Bayes (NB) and BERT for story emotion classification. . . . .	103
6.3	Accuracy in % of both the GPT-2 and LSTM models for music emotion classification. . . . .	105
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). . . . .	107
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. . . . .	120
7.2	Accuracy of each model in conveying the target emotions E0, E1, E2 and E3. . . . .	121

## **Abstract**

### Controlling Neural Language Models for Affective Music Composition

by

Lucas N. Ferreira

Deep generative models are currently the leading methods for algorithmic music composition. However, one of the major problems of these methods 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 fine-tune 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 in 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 3,640 unlabelled pieces and 200 ones labeled according to the circumplex model of emotion. The three methods were evaluated with listening tests, in which human subjects indicated that the methods could convey different target emotions.



# Chapter 1

## Introduction

Music composers have been using algorithms, rules, and general frameworks for centuries as part of their creative process to compose music [96]. 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 [96]. Lejaren Hiller and Leonard Isaacson wrote the program that generated this composition with an ILLIAC computer at the University of Illinois [54]. Since then, many different methods

have been proposed to generate music with computers: expert systems [31], generative grammars [69], cellular automata [6], evolutionary algorithms [59], Markov chains [17], and neural networks [131]. 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 [36] and supported applications in music analysis [82], procedural audio [38], audio synthesis [35], music therapy [138], 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) [46]. Consequently, AMC researchers started exploring different types of neural networks to generate music: recurrent neural networks (RNNs) [100], transformers [60], convolutional neural networks (CNNs) [61], variational autoencoders (VAEs) [115], generative adversarial networks (GANs) [29], 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 words in a language [8]. Modern neural LMs compute the conditional probability of a token  $x^t$  given prefix tokens  $x^1, x^2, \dots, x^{t-1}$  by first computing a dense vector representation of the prefix and then feeding it into a classifier to predict the next token [127]. 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  $x^1, x^2, \dots, x^t$  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, 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) [15]. Modern musical LMs have been capable of generating high quality pieces of different styles with strong short-term dependencies<sup>1</sup> [60].

Transformers and RNNs can learn a LM by processing input sequences of tokens  $x^1, x^2, \dots, x^t$  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  $\hat{y}^t$  distribution, 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 LSTMs [133]. One drawback of the transformers is that they can only process sentences with a fixed size instead of LSTMs that can process sentences of any size.

---

<sup>1</sup>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 [40]. 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 [137]. Applications involve generating soundtracks for movies and video games [136], sonification of biophysical data [20], and generating responsive music for the purposes of music therapy and palliative care [91].

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 (e.g. tempo, melody, harmony, rhythm, timbre, dynamics) evoke emotions. However, due to the large space of features, 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. [109], the first explored approach is a genetic algorithm that optimizes the neurons of  $L$  that carry sentiment signal (positive or negative)<sup>2</sup>, 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 [118]. 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

---

<sup>2</sup>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) [40].

### 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 [112] and 2,065 were scraped from publicly available sources on the internet. The LM  $L$  was implemented with a GPT2 [111] 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 [102] and 16th [39] 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* [60] 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 these chapters.

## 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), in his work *Micrologus*, describes a method for mapping Latin lyrics into melodies. The method extracts the vowels from given Latin lyrics and then maps the vowels into pitches. Figure 2.1 shows this procedure considering the seven different natural pitches of the western music system<sup>1</sup>. 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 this verse from the *Hymn to St John* with the extracted vowels in parenthesis:

REsonare fibris (eoaeii)

MIra gestorum (iaeou)

FAMuli tuorum (auiou)

SOLve polluti (oeoui)

LABii reatum (aiieau)

Sancte Ioannes (aeioae)

Using the mapping rules defined in Figure 2.1, the first line of the verse can be mapped to the melody *D, B, F, C, C, E*.

Around 1280, the music theorist Franco de Cologne, in his work *Cantus Mensuralis*, introduced a music notation system where the note durations are defined by their shapes. This new notation allowed composers to treat rhythm independently of the pitch. For example,

---

<sup>1</sup>C, D, E, F, G, A, B

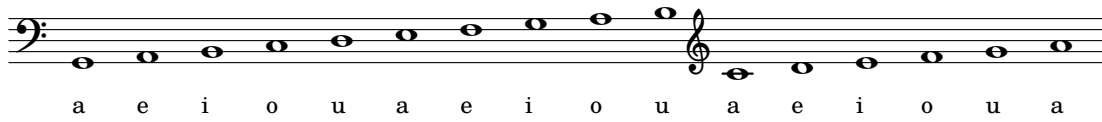


Figure 2.1: Mapping of vowels on tone pitches by Guido of Arezzo.

French composers of the *ars nova*, such as Phillipe de Vitry and Guillaume de Machaut, used a technique called isorhythm to map a rhythmic pattern (named the *talea*) onto a pitch contour (named the *color*). Figure 2.2 shows the tenor melody of the piece *De bon espoir-Puisque la douce-Speravi* by Guillaume de Machaut built using the isorhythm technique.

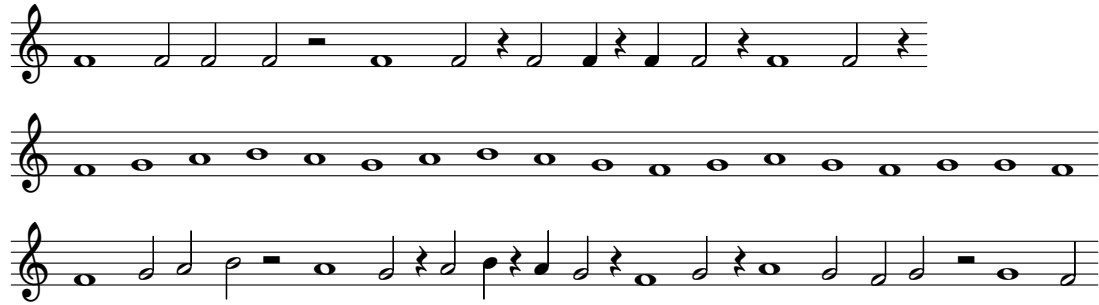


Figure 2.2: The tenor of *De bon espoir-Puisque la douce-Speravi* by Guillaume de Machaut.

The first staff is the *talea* and the second one is the *color*. The third staff is the resulting piece 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. Fugues and canons are highly procedural forms of contrapuntal compositional techniques. Counterpoint consists of interleaving two or more melodies that are harmonically dependent but independent in rhythm and melodic contour. 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<sup>2</sup>, augmentation<sup>3</sup>, or inversion<sup>4</sup> [122]. Figure 2.3 shows the first twelve measures of the canon named *Canon per Augmentationem in Contrario Motu* from *The Art of Fugue*.

Figure 2.3 displays the first twelve measures of the canon *Canon per Augmentationem in Contrario Motu* from *The Art of the Fugue* by Johann Sebastian Bach. The score is presented in two systems. The first system shows the initial entry of the leader (voice 1) and the follower (voice 2) in measure 5. The second system continues the canon. Harmonic analysis is provided below the staves, indicating the key and mode for each measure.

First system analysis:

- Measure 1: D minor: Re menor: i
- Measure 2: V<sup>5</sup>
- Measure 3: i
- Measure 4: G minor: Sol menor: IV<sup>5</sup>
- Measure 5: vii<sup>7</sup>
- Measure 6: i

Second system analysis:

- Measure 7: Bb major: Sib mayor: V<sup>5</sup>
- Measure 8: I
- Measure 9: V<sup>6</sup>
- Measure 10: D minor: Re menor: vii<sup>4</sup>
- Measure 11: i
- Measure 12: ii
- Measure 13: vii<sup>6</sup>
- Measure 14: i<sup>6</sup>
- Measure 15: vii<sup>4</sup>
- Measure 16: i

Figure 2.3: *Canon per Augmentationem in Contrario Motu* from *The Art of the Fugue* by Johann Sebastian Bach.

Figure 2.4 shows the follower voice (measures 5 to 13) transforming the leader (measures 1 to 5) using augmentation and inversion. The follower voice, due to augmentation, needs eight measures to answer the first four measures.

In the Classical Period (1750-1827), *Musikalisches Würfelspie* (German for a musi-

<sup>2</sup>Moving a set of notes up or down in pitch by a constant interval

<sup>3</sup>Repeating a set of notes with longer durations.

<sup>4</sup>Playing a given set of notes upside down, reversing the contour of the notes.

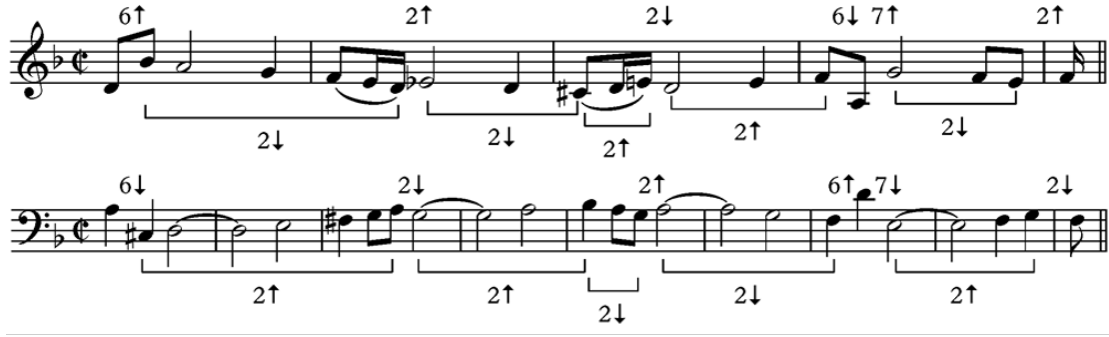


Figure 2.4: Subject of the canon.

cal 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 [21]. Mozart's dice game was designed to generate sixteen-measure-long minuets<sup>5</sup>. 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. 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 the 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 the Table 2.1 is then used to retrieve a single measure from a collection of musical fragments. Figure 2.5 shows the first eight measures of a minuet that can be generated using the Table 2.1.

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

<sup>5</sup>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 pre-composed measure.



Figure 2.5: Example of minuet generated using Mozart's dice game.

with extensive use of chromaticism<sup>6</sup>. 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 (or row) of musical elements (such as pitches, rhythms, dynamics, timbres, and others) into a pattern that repeats itself throughout a composition.

A basic form of serial composition consists of selecting a given number of notes on

<sup>6</sup>Chromaticism is the use of notes outside the scale of which a composition is based.

the chromatic scale<sup>7</sup> 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<sup>8</sup> or retrograde inversion<sup>9</sup> can be applied to the tone to introduce variation into a serial composition. *Twelve-tone serialism* is a serial technique where the tone row is an ordered arrangement of 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 2.6 illustrates the tone row used by Alban Berg in the *Lyric Suite* for string quartet composed in 1926.



Figure 2.6: Tone row used by Alban Berg in the *Lyric Suite* for string quartet composed in 1926.

Table 2.2 shows different transformations that can be applied to the tone row showed in Figure 2.6. 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,  $T_5$  means the tone row has been transposed up five half steps from the original form

<sup>7</sup>A musical scale with twelve pitches, each a semitone, above or below its adjacent pitches.

<sup>8</sup>Playing a sequence of notes backwards.

<sup>9</sup>Playing a sequence of notes backwards and upside down.

(e.g. a Perfect Fourth).

	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 2.6.

In the 20th century, Iannis Xenakis deeply explored the use of statistical methods to compose *stochastic music* [141] – 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<sup>10</sup>. In *Achorripsis* (1957), he used Poisson’s distribution of rare events to organize “clouds” of sound [2]. John Cage is another important composer of 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.

<sup>10</sup>Continuous transition between two notes of different pitches.

## 2.2 Computational Methods

The *ILLIAC Suite: String Quarter No 4* [54], 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, density) and the note parameters (e.g. pitch, duration) of his compositions [86]. *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 multi-media piece called *HPSCHD*. HPSCHD used a dice game approach where the pre-composed snippets of music were extracted from pieces from 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 which was 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 [43], generative grammars [21], cellular automata [92], evolutionary algorithms [59], Markov chains [53], neural networks [131], and others. The remainder of this chapter briefly introduces these methods, except neural networks, which are discussed in great 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 [43] 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 as a *constraint satisfaction problem* (CSP) [4]. For example, Ebcioglu [31] 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<sup>11</sup>. Expert systems can also be formalized with *case-based reasoning*. Pereira

---

<sup>11</sup>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.

et al. [106] 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 [58]. The expansion rules can be manually defined or inferred from analyzing a corpus of pre-existing music compositions. Lidov and Gadura [84] presented an early example of generative grammar manually designed for the generation of melodies with different rhythmic patterns. A more recent example is the work of Keller and Morrison [69], 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) [21], which automatically derives a special type of grammar called *Augmented Transition Network* from a corpus of compositions in a specific style. EMI extracts this augmented transition network by finding short musical patterns that 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 [139]. In his 1986 work *Horos*, Xenakis designed a CA to produce harmonic progressions and new instrument combinations [126]. CAMUS [92] is a system that combined two bi-dimensional CAs to compose polyphonic music: Conway's *Game of Life* [42] and Griffeaths *Crystalline Growths* [25]. Each activated cell in the Game of Life was mapped to a *triad* *A tuple of three notes*., whose instrument was selected according to the corresponding cell in the Crystalline Growths CA. WolframTones [6] 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, jazz) and set different parameters of the algorithm (e.g. rule number, rule type, seed). Moreover, users can define how to map the CA patterns to different musical features (e.g. pitch, tempo, timber).

### 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 a 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 current. Horner and Goldberg [59] presented one of the first examples of EA for music composition. Their algorithm was inspired by a composition technique called *thematic bridg-*

*ing*, where the beginning and end of a piece are given, and a fixed number of transformations (e.g. transposition, inversion, retrograde, etc) 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 are the works of McIntyre [89] for four-part baroque harmonization, Polito et al. [107] for counterpoint composition, and Papadopoulos and Wiggins [104] 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 musical Interactive Genetic Algorithms (IGAs). For example, GenJam [12] 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 are presented in the works of Jacob [63], Schmidl [119], Tokui and Iba [132], and many others.

### **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 transi-

tions between successive states [44]. In a first-order Markov chain, the probability of the next state depends only on the current state, but in an  $n$ th-order Markov chain, the probability is conditioned on the previous  $n-1$  states [44]. 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 2.7 shows an example of a simple abstract Markov Chain.

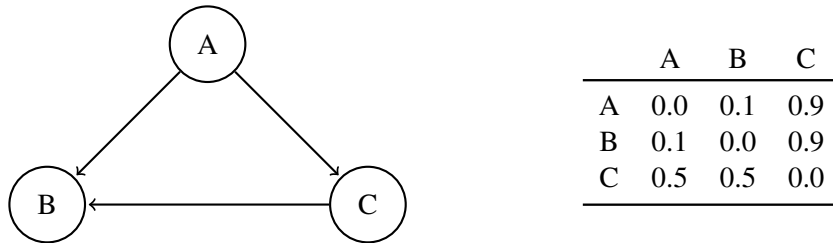


Figure 2.7: Example of Markov Chain represented as a directed.

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. [17] presented one of the first examples inferred from a corpus of music. Brooks et al. experimented with different orders

of Markov chains where each state represents a pitch class. The probability tables of these chains were learned from thirty-seven common meter ( $\frac{4}{4}$ ) hymn tunes (monophonic). While Markov chains designed manually by composers worked well for specific compositional tasks [130, 67, 78], those learned from a corpus, in practice, can capture only short-term dependencies in music [94]. 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 [94].

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 [85]. 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 of methods presented in this section require musical models (or rules) to be defined and encoded to manipulate or evaluate music. Although music theory formalizes several aspects of music analysis and composition, defining and encoding computational music models for AMC is still very challenging, given that music composition is a creative task. Some of the presented methods, such as generative grammars and Markov models, can infer these rules from a corpus of music. However, in practice, they have shown to 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 strong 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) [81]. These results also motivated AMC researchers to explore deep learning algorithms for music composition [15]. The next chapter presents a detailed discussion on how deep learning can be used to learn music rules from a corpus of pieces.

## 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, video, etc) [46]. As ML algorithms, deep neural networks are used to learn different tasks from examples without being explicitly programmed to do so, including *supervised learning* tasks, *unsupervised learning* tasks, and *reinforcement learning* tasks. In a *supervised learning* task, pairs  $(x^i, y^i)$  of inputs  $x^i$  and target classes (also called labels)  $y^i$  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  $x^i$  can have one of two possible labels  $y^i = \{0, 1\}$ . In multiclass problems,

the label  $y^i = \{0, 1, \dots, L\}$ . can take one of  $L > 2$  values. A classic example of a binary problem is email spam detection (spam or not). A popular example of a multiclass problem is classifying images of handwritten digits [80]. In an unsupervised learning problem, only the inputs  $x^i$  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 [9]. In reinforcement learning, the NN is trained to learn an agent that can take optimal actions in an environment according to a *reward function*.

Most modern music composition systems are designed as unsupervised learning tasks, where a deep NN has to learn relationships from music structures (e.g. notes, chords, 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, music generative models are typically designed as a *neural language model* (LM). In NLP, a LM is a conditional probability  $P(x_t | x_1, \dots, x_{t-2}, x_{t-1})$  of the next word  $x_t$  given a context with the  $t$  previous words  $\{x_1, \dots, x_{t-2}, x_{t-1}\}$  of a sentence. One can create a LM  $P$  from examples  $(x_t, x_{t+1})$  by training a NN to predict the next word  $x_{t+1}$  from the current word  $x_t$ . A NN trained this way can generate new sentences by sampling from the learned distribution  $P$ . Considering that music is a sequence of musical “words” (e.g., notes, chords, sections), one can train a neural LM to compose music by (a) creating a dataset of symbolic music, (b) defining the architecture of the NN to learn the conditional probability  $P$ , and (c) sampling tokens from the learned conditional probability  $P$ . 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 such as 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. 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 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 either the time of the note event in a 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, i.e. 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.

### 3.1.2 Piano Roll

Another very common format of symbolic music is the *piano roll*. This format 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 modern piano roll representation.

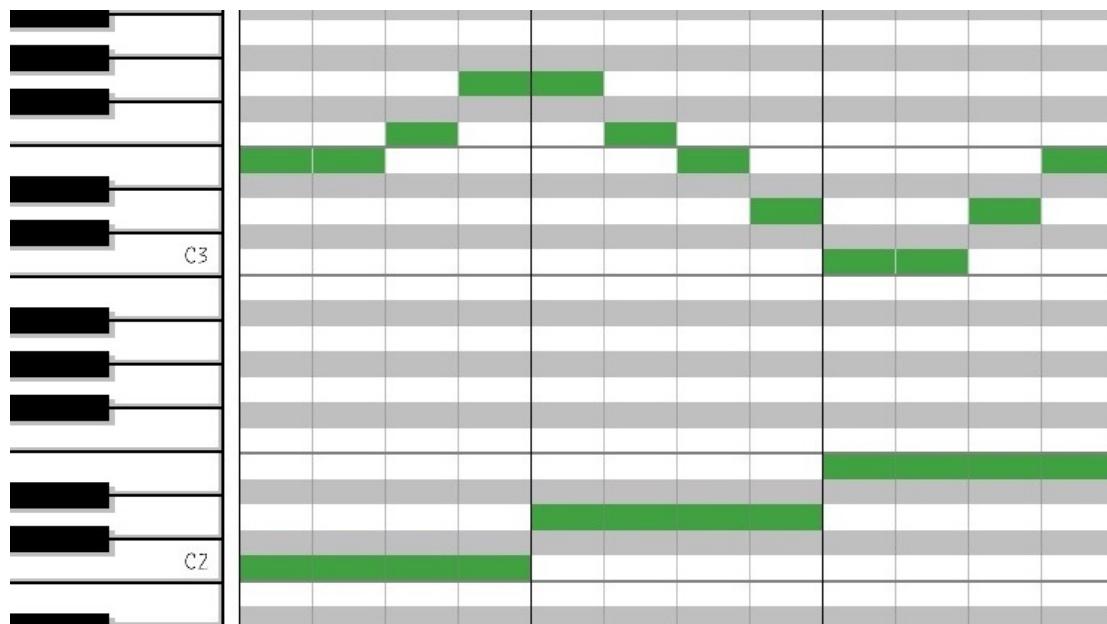


Figure 3.1: Example of music represented in a piano roll format.

The MIDI representation can be mapped to a piano roll by sampling time steps with a given frequency (e.g. every eighth note). Because of this property, most datasets of symbolic music organize pieces in a collection of MIDI files. For example, the *MAESTRO* dataset [50] 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* [112] 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* [11]. *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 [13] 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, there are also other formats that have been used in the AMC literature. For example, the ABC notation [135] is a text-based music notation system popular for transcribing, publishing and sharing folk music. MusicXML [45] 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 music symbols. These symbols will be used to create sequences that represent music pieces. 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 symbols that will form the music pieces. 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 symbol in the vocabulary has to be mapped into a vector. Traditionally, these symbols are mapped using *one-hot* encoding, where each symbol is given an index  $i$  and is represented by a vector  $v = [v_1, v_2, \dots, v_n]$  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 symbols in the vocabulary.

## 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 neurons 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  $x = [x_1, x_2, \dots, x_n]$  of incoming connections from the previous layer and assigns a weight vector  $w = [w_0, w_1, \dots, w_n]$  to these connections. In its most basic form, the neuron first applies a linear transformation  $z = wx + b$  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<sup>1</sup> NN called *feedforward* network or *multilayer perceptron* (MLP), with the computation performed by each layer highlighted besides them. The bias terms have been removed for clarity.

The number of neurons per layer, i.e. the *layer size*, and the layers' activation functions depend on the task that the NN is learning. 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 (zero to nine). 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 size

---

<sup>1</sup>Typically, the input layer is not considered when counting the depth of the NN.

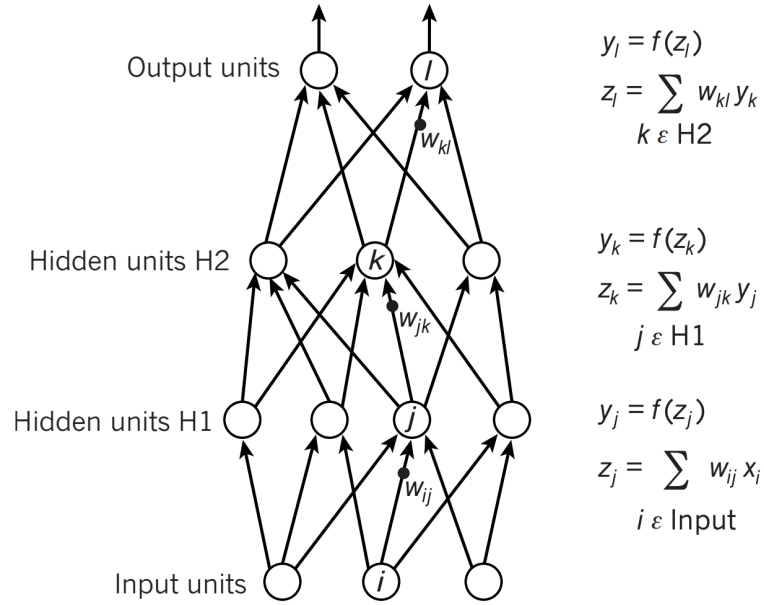


Figure 3.2: Diagram of a feedforward neural network [81].

of the hidden layers is defined arbitrarily and should be controlled to optimize the performance (e.g. prediction accuracy) of the network.

In *multiclass* problems, such as the handwritten 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 a single output  $0 \leq \hat{y} \leq 1$ . The NN predicts 1 if  $\hat{y} > 0.5$  and 0 otherwise. The activation functions used in the hidden layers are decided arbitrarily as well, 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 formally describes each of these functions.

NNs are typically trained with some variation of the *gradient descent* (GD) algorithm,

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)	$\text{relu}(x) = \max(0, x)$
Softmax	$\text{softmax}(x) = \frac{e^{x_i}}{\sum_{j=1}^J e^{x_j}}$

Table 3.2: A list of common activation functions in the hidden layer.

which optimizes the weights  $W^i$  and  $b^i$  of all layers  $i$  of the network to minimize a given *loss function*  $J(W, b)$ . The loss function depends on the task being modeled by the NN. 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 the NN. 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 the  $NN_{W^i, b^i}$  with parameters  $W$  and  $b$  for class  $c$ , and  $C$  is the number of classes in the prediction task.

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

As shown in Algorithm 1, GD works by iteratively taking steps in the opposite direction of the gradient of the loss function with respect to all weights. For a given number of iterations called *epochs*, GD (line 2) computes the gradient vector of the loss function  $J(W, b)$  for the entire training dataset and (line 3) updates the parameters  $W^i$  and  $b^i$  in the opposite direction of the gradient. The *learning rate*  $\alpha$  is a parameter that controls the size of the training step. Computing the gradient vector (line 2) requires calculating the partial derivatives of the loss function with respect to all the weights in the NN. This calculation is typically performed by an algorithm called *backpropagation*, which uses the chain rule to compute the gradient one

---

**Algorithm 1** Gradient Descent

---

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

**Ensure:** Updated parameters  $W^i$  and  $b^i$  of all layers  $i$ .

```
1: for  $i \leftarrow 1$  to  $e$  do  
2:    $\partial W^i \leftarrow \frac{\partial J}{\partial W^i}, \partial b^i \leftarrow \frac{\partial J}{\partial b^i}$   
3:    $W^i \leftarrow W^i - \alpha \partial W^i, b^i \leftarrow b^i - \alpha \partial b^i$   
4: end for
```

---

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* of size  $N$  and performing a parameter update for each one of them. Although SGD supports training with 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 per-parameter and separately adapting them during training [74]. In practice, most practitioners use the Adam optimizer, given that successful NNs typically require large datasets that do not fit in memory<sup>2</sup>.

---

<sup>2</sup>The term *big data* is typically used to refer to these very large datasets.

### 3.2.1 Recurrent Neural Networks

*Recurrent Neural Networks* (RNNs) are an important architecture for AMC because they were specifically designed to model sequential data. RNNs process sequences  $x = [x_1, x_2, \dots, x_t]$  step-by-step by keeping an internal state  $h_t$  that is updated every step. Each element  $x_i$  is a vector representing 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 an RNN.

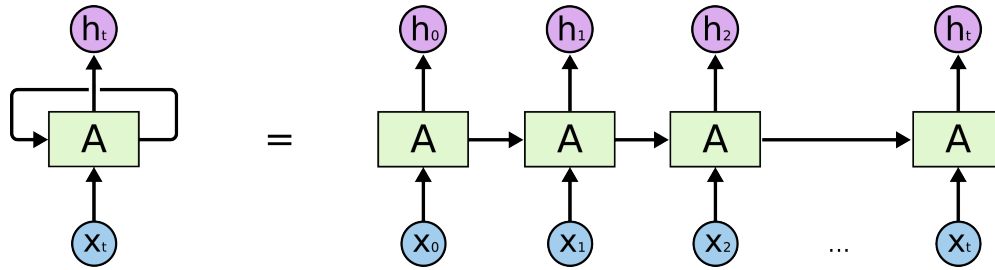


Figure 3.3: Diagram of a RNN [98].

On the left-hand side, the RNN is shown with an input layer that passes a symbol  $x_t$  to a stack of hidden layers  $A$  that updates  $h_t$ . A loop in the hidden layers 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 multiple outputs, one per time step or one single output at the very last time step. This configuration depends on the problem. However, the output layer maps a current hidden state  $h_t$  to an output vector  $y$ .

Two of the most simple types of RNNs are the *Elman* and *Jordan* networks. They are RNNs with three layers: an input layer, a single hidden layer, and an output layer. In these



networks, an output  $y_t$  is produced for each time step  $t$ . Equation 3.2 and Equation 3.3 formally describe these networks, respectively, where  $W_{xh}$ ,  $W_{hh}$  and  $W_{hy}$  are weight matrices and  $\sigma$  are activation functions. The matrices  $W_{xh}$ ,  $W_{hh}$  represent the weights of the hidden layer, and the matrix  $W_{hy}$  represent the weights of the output layer. The weights  $W_{hh}$  are called context units.

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1}) \quad (3.2)$$

$$y_t = W_{hy}h_t$$

$$h_t = \sigma(W_{xh}x_t + W_{hh}y_{t-1}) \quad (3.3)$$

$$y_t = W_{hy}h_t$$

These two networks are very similar. The main difference is that the context units are fed from the output layer instead of the hidden layer. Todd [131] presented one of the first applications of NNs for music composition: a *Jordan* network designed to generate melodies (monophonic). 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 could give new melodies as input to the network, which would create new melodies interpolating between the ones used during the training phase. Duff [30] published another early example that is similar to the work of Todd [131], but Duff encoded each melody as a sequence of note intervals<sup>3</sup> instead of pitch classes.

---

<sup>3</sup>The distance in pitch between two notes.

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 an 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 [7]. 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) network [55] is a special type of RNN explicitly designed to solve the problem of simple RNNs with long-term dependencies. As shown in Figure 3.3, RNNs have the form of a chain of repeating stacks of layers  $A$ . LSTMs follow the same structure, but the repeating module performs different computations. Figure 3.4 shows a diagram of a LSTM. 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. The flow of information in the cell state is controlled by four *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 3.4 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  $h_{t-1}$

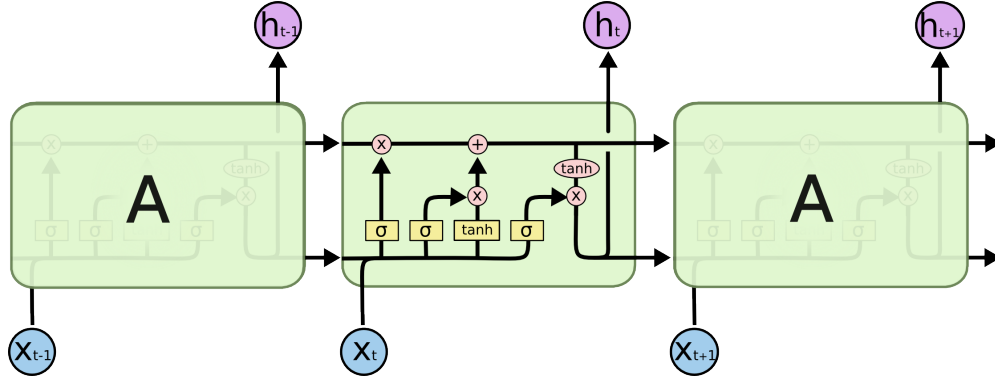


Figure 3.4: Diagram of a LSTM [98].

concatenated with the input vector  $x_t$ , and  $\tilde{C}_t$  is the candidate vector to added to the cell state.

$$\begin{aligned}
 f_t &= \sigma(W_f[h_{t-1}, x_t]) \\
 i_t &= \sigma(W_i[h_{t-1}, x_t]) \\
 \tilde{C}_t &= \tanh(W_c[h_{t-1}, x_t]) \\
 C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
 o_t &= \sigma(W_o[h_{t-1}, x_t]) \\
 h_t &= o_t * \tanh(C_t)
 \end{aligned} \tag{3.4}$$

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 hidden layer that is added before any other hidden layer to transform the one-hot input vectors  $x_t$  into *embeddings*. In NLP, a word *embedding* is

a learned representation for text where words that have the same meaning have a similar representation. The *embedding* layer learns a dense vector representation of arbitrary dimension from the sparse one-hot representation of the symbols in the vocabulary.

Early RNNs were not necessarily trained as LMs (see Section 3.2.1), but most modern LSTMs compose music as a LM. To learn a LM from an LSTM, one has to map the output at each time step  $\hat{y}_t$  to a probability distribution over the symbols in the defined music vocabulary. This is normally done with a *softmax* activation function in the output layer. The LSTM is then trained with mini-batch gradient descent to predict the next symbol  $y_{t+1}$  given the current symbol  $y_t$  from music pieces given as training data. Many recent deep learning approaches used LSTMs to model music generation with LMs [15]. For example, Lyu et. al. [87] combined an LSTM with another type of neural network called Restricted Boltzman Machine (RBM) to generate polyphonic music with different musical styles. They experimented with both the Musedata and the JSB Chorales [13]. The pieces were encoded as a sequence of pitch numbers temporally aligned on an integer fraction of a quarter note. This encoding system only considered pitch and duration and hence did not considered dynamics information (e.g. note velocity).

BachBot [83] uses a deep LSTM to generate polyphonic in the style of Bachs chorales. BachBot was trained with the JSB Chorales dataset, where each chorale was encoded with a sequence of sixteenth-note frames. Each frame consisted of four tuples (Soprano, Alto, Tenor, and Bass), and each tuple in the form (*pitch*, *tie*), where *pitch* represents a MIDI pitch and *tie* is a boolean value that distinguishes whether a note is tied with a note at the same pitch from the previous frame or is articulated at the current time step. This encoding system also did not consider dynamics information. Oore et al. [100] propose one of the first methods that can generate

music with dynamics. They trained an LSTM with a new representation that supports tempo and velocity events from MIDI files. This model was trained on the MAESTRO [50]. With this new representation and dataset, Oore et al. [100] generated more human-like performances when compared to previous models.

### 3.2.3 Transformers

*Transformer* [133] is a modern architecture for sequence modeling based on *attention* mechanisms, as opposed to recurrent layers. In NLP, an attention mechanism is a part of a neural architecture that enables highlighting relevant words of the input sequence dynamically. Instead of keeping an internal hidden state that is updated at each time step like RNNs, transformers process entire sequences associating an *attention* score to each token that 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 transformers can be trained considerably faster than LSTMs [133]. One drawback of the 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 (French). Machine translation is a *sequence-to-sequence* problem, where a sequence input  $x$  has to be 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 as one-hot vectors and transforms it into two sequences: a sequence of *word embeddings* and a sequence of *positional embeddings*. The former is a dense vector representation of words learned from the sparse one-hot input. The latter is also a dense vector representation but learned from the position (indices) of the words in the input sentence. The transformer adds the input embeddings and positional encodings together and passes the result through the encoder, followed by the decoder.

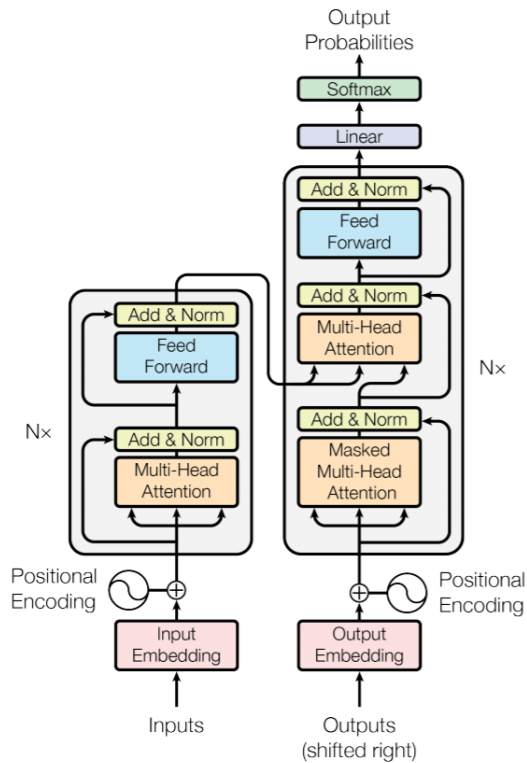


Figure 3.5: Diagram of a Transformer [133].

The encoder converts the (input + position) embeddings  $e$  into *encodings* with 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 [51] is applied around each of the two layers, followed by a layer normalization [5]. A residual connection is a connection between non-contiguous layers. A layer normalization is an extra layer that 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.

The most important part of the transformer architecture is the *multi-head attention layer*, which computes a score matrix  $Z$  from the embeddings  $e$ , where 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.

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 neural network is the scaled dot-product attention, which is shown in Equation 3.5. The matrices  $Q$ ,  $K$ , and  $V$  are created by packing the embeddings  $e$  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.

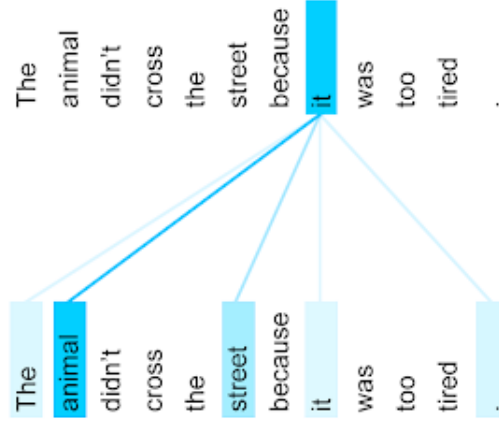


Table 3.3: The encoder self-attention distribution for the word “it” in the sentence “The animal didn’t cross the street because it was too tired.” [133].

$$Z = \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$Q = E * W_q \tag{3.5}$$

$$K = E * W_k$$

$$V = E * W_v$$

The decoder is similar to the encoder, but it has an extra attention layer added before the other two to perform multi-head attention over the target output sentence  $y$ . This extra layer uses a mask to ensure that the predictions for position  $i$  can depend only on the known outputs at positions less than  $i$ . The output of the decode is a linear layer followed by a softmax activation.

One can take the decoder portion of the transformer to train a LM capable of generating sequences such as text or music similar to a LSTM LM (see Section 3.2.2). Equation 3.6 formally defines a decoder-based transformer LM, where  $x = [x_1, x_2, \dots, x_t]$  is the input se-



quence,  $n$  is the number of layers,  $W_i$  is input embedding weight matrix, and  $W_p$  is the position embedding weight matrix.

$$\begin{aligned}
h_0 &= W_i x + W_p \\
h_l &= \text{transformer\_block}(h_{l-1}) \forall i \in [1, n] \\
P(u) &= \text{softmax}(h_n W_e^T)
\end{aligned} \tag{3.6}$$

Transformers are currently the state-of-the-art of both natural and music language modeling. Radford et. al. [110, 111] proposed a series of models called GPT (General Pre-trained Transformer), GPT-2, and GPT-3 that used only the decoder part of the original transformer to create a model of natural language. Radford et. al. showed that besides generating long coherent sequences of text, pre-trained GPT models can be fine-tuned with a simple extra linear layer to perform specific NLP tasks (e.g. commonsense reasoning, question answering, summarization, and others) with state-of-the-art performance. 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 adding an extra linear layer to the pre-trained transformer and training this layer with a smaller dataset explicitly created for the task (supervised).

Music Transformer [60] is one of the first decoder-transformer to generate music. It uses a new relative attention mechanism that improves memory consumption of the original decoder, and hence it can process longer sequences. Music Transformer achieves state-of-the-art performance on the MAESTRO dataset [50]. Donahue et. al. [28] showed that a decoder-transformer could also compose multi-instrument scores by training it with the NES MDB [27] dataset. Donahue et al. improved the performance of their model with a pre-training procedure

similar to Radford et al. [110]. The transformer was first pre-trained with the diverse Lakh dataset (multiple instruments) and then fine-tuned with NES-MDB (four instruments). They manually defined a mapping between the instruments from the two datasets. Pop Music Transformer [62] 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) [75] 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 any data. 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.

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 have fewer dimensions than the input data itself, 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.

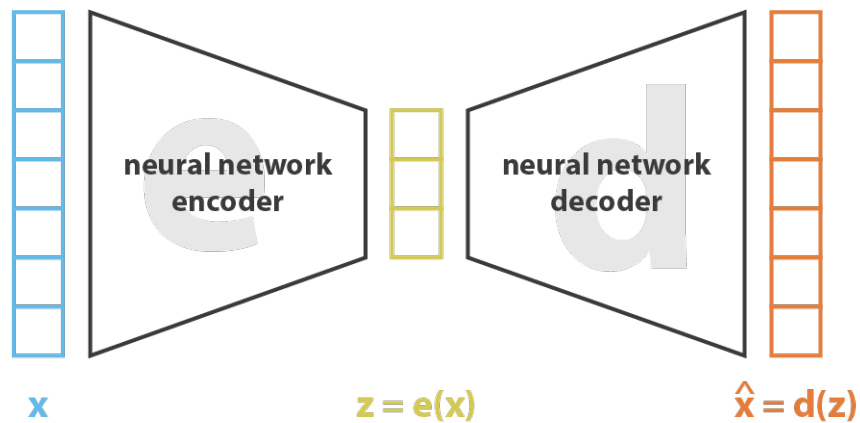


Figure 3.6: Diagram of an Autoencoder [116].

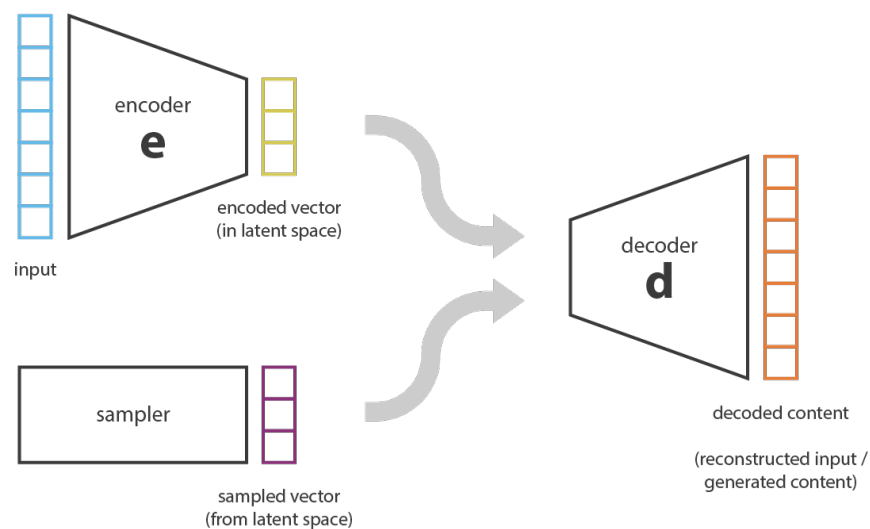


Figure 3.7: Sampling from the latent space built by an Autoencoder [116].

One limitation of the autoencoder is that it often learns a latent space that is not continuous<sup>4</sup> nor complete<sup>5</sup>. This means that if one decodes a random vector sampled from the

<sup>4</sup>Two close points in the latent space should yield similar outputs when decoded.

<sup>5</sup>Any point sampled from the latent space should yield a “meaningful” output when decoded

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 autoencoders and VAEs is that the encoder of the VAE outputs two vectors  $\mu_x$  and  $\sigma_x$ , instead of a single vector  $z$ . These two vectors represent the mean and standard deviation of a normal distribution  $N$ . The decoder samples a vector  $z \sim N(\mu_x, \sigma_x)$  from the distribution  $N$  and reconstructs the input  $x$  from the sampled  $z$  using the decoder  $d$ .

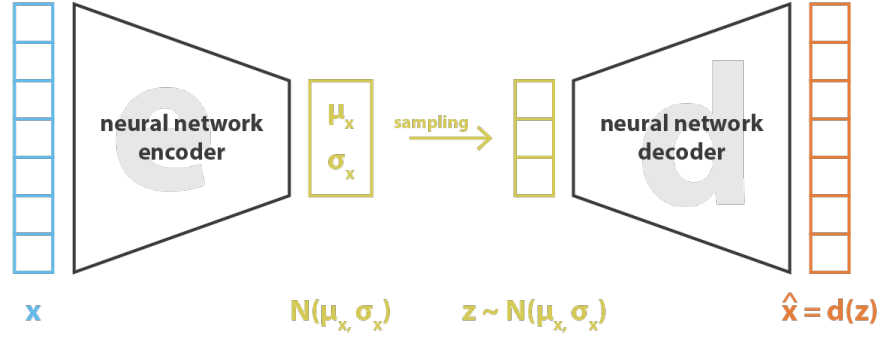


Figure 3.8: Diagram of a Variational Autoencoder (VAE) [116].

MusicVAE [114] is one of the first examples of VAE applied to music generation. It starts by splitting the input sequence  $x$  into  $U$  non-overlapping subsequences  $y_u$ , such that  $x = [y_1, y_2, \dots, y_U]$ . The encoder processes the segmented input  $x$  with a bidirectional LSTM network that outputs two hidden states  $\overleftarrow{h}_t$  and  $\overrightarrow{h}_t$  that are concatenated to form a final hidden state  $h_t$ . The final hidden state  $h_t$  is fed into two fully connected layers to produce the latent distribution parameters  $\mu$  and  $\sigma$ . The decoder is a novel hierarchical LSTM that takes the latent

vector  $z \sim N(\mu_x, \sigma_x)$  as input and first produces  $U$  embedding vectors  $c = [c_1, c_2, \dots, c_U]$ , one for each subsequence  $y_u$ . An LSTM called the *conductor* produces these embedding vectors, where the hidden state is initialized by passing  $z$  through a fully connected layer followed by a  $\tanh$  activation function. Once the conductor has produced the sequence of embedding vectors  $c$ , each one is individually passed through a shared fully connected layer followed by a  $\tanh$  activation to produce initial states for a final decoder LSTM. The decoder RNN then autoregressively produces a sequence of distributions over output tokens for each subsequence  $y_u$  via a softmax activation in the output layer. At each step of the final LSTM, the current conductor embedding  $c_u$  is concatenated with the previous output token to be used as the input.

MIDI-VAE [18] uses a VAE to perform style transfer in polyphonic symbolic music. It works by separating a portion of the latent vector  $z$  for style classification  $z_s$ , and another part  $z_t$  to encode the characteristics of the music dataset. During generation, one can change the style  $S_i$  of a given input piece  $x$  to another style  $S_j$  by passing a given input piece through the encoder to get  $z$  and swapping the values of dimensions  $z_s^i$  and  $z_s^j$ , and passing the modified latent vector through the decoder. Style labels can be music genres such as Jazz, Pop, and Classic; or composer names such as Bach or Mozart. VirtuosoNet [65] is a VAE designed to generate piano performances with expressive control of tempo, dynamics, and articulations. The encoder is a hierarchical LSTM such that the input music is encoded on different levels: note, beat, and measure. The decoder renders musical expressions consistently over long-term sections by first predicting the tempo and dynamics in measure level and, based on the result, refining them in note level.

### 3.2.5 Generative Adversarial Networks

Similar to VAEs, *Generative Adversarial Networks* (GANs) [48] are powerful 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 then transforms this noise into a fake example. The discriminator is a binary classifier that discriminates examples as *fake* (0) or *real* (1). The generator and the discriminator architectures depend on the data type (e.g. text, images, videos) that is being modeled. For example, Figure 3.9 illustrates a GAN for handwritten digit generation. In this case, the random noise is a matrix  $M$  representing a grayscale image. A simple way to generate such a matrix is to sample grayscale values between 0 (black) and 1 (white) from a uniform distribution. The generator transforms the random matrix  $X$  into a fake handwritten digit normally using convolutional layers. The discriminator combines real and fake images to learn how to separate them between fake and real.

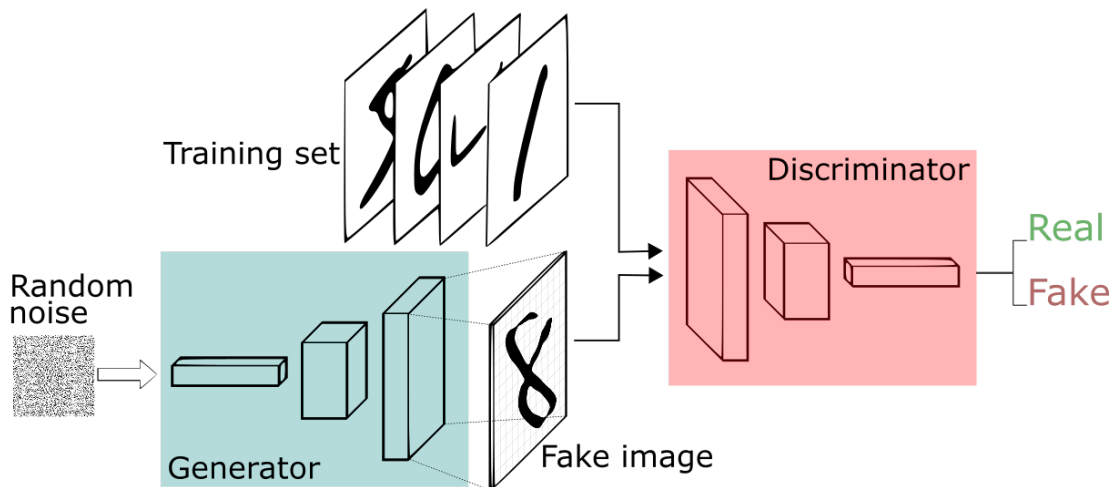


Figure 3.9: Diagram of a Generative Adversarial Network (GAN) [121].

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 the discriminator 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.
2. The generator trains for one or more epochs. The generator is trained with a loss function that penalizes the generator for producing a sample that the discriminator network 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. It's important to highlight that only the generator weights are updated during generator training.

The original GAN uses a single loss function called *minimax loss* to train both the generator and the discriminator. Minimax loss is derived from the cross-entropy loss between the real and generated distributions. The generator tries to minimize this loss while the discriminator tries to maximize it. Equation 3.7 formally defines the minimax loss, where  $D(x)$  is the discriminator's estimate of the probability that real data instance  $x$  is real,  $E_x$  is the expected value over all real data instances,  $G(z)$  is the generator's output when given noise  $z$ ,  $D(G(z))$  is the discriminator's estimate of the probability that a fake instance is real, and  $E_z$  is the expected value over all generated fake instances  $G(z)$ .

$$E_x[\log(D(x))] + E_z[\log(1 - D(G(z)))] \quad (3.7)$$

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  $G$ . This approach works well to generate continuous data such as images [16], 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 by a discriminator. 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 [143] generated polyphonic music as a series of note events by introducing some ordering of notes and using RNNs in both the generator and the discriminator. SeqGAN [146] combined GANs and reinforcement learning to generate monophonic music with an RNN generator and a CNN discriminator. MidiNet



[143] used GANs with conditional CNNs as both generator and the discriminator to generate melodies that follow a chord sequence given a priori, either from scratch or conditioned on the melody of previous bars. MuseGAN [29] generates polyphonic music by encoding multi-track measures as a piano roll and also modeling both the generator and the discriminator as CNNs. Muhamed et al. [95] modeled both generator and the discriminator as transformers and used the Gumbel-Softmax trick [64] 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 architectures specially designed to model sequences: RNNs, LSTMs, and Transformers. Formally, these sequential models use a softmax activation function in the output layer to create a LM  $L = P(x_t | x_1, \dots, x_{t-2}, x_{t-1})$ , where  $\{x_1, \dots, x_{t-2}, x_{t-1}\}$  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. Namely, one starts with a prior sequence of tokens  $x = [x_1, x_2, \dots, x_{t_1}]$ , which are 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 symbol is found or a given number of symbols are generated. As defined in Equation 3.8, autoregressive generation assumes that the probability distribution of a sequence of tokens can be decomposed into the product of conditional next token distributions.

$$P(x_t | x_1, \dots, x_{t-2}, x_{t-1}) = \prod_{t=1}^T P(x_t | x_{1:t-1}) \quad (3.8)$$

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 as music composition, searching fits better 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 an example where the prior is the sequence  $x = [The]$ . In the first step, the word *car* is sampled from the conditional probability distribution  $P(x_1|The)$  and, in the second step, the word *drives* is sampled from  $P(x_2|The, car)$ .

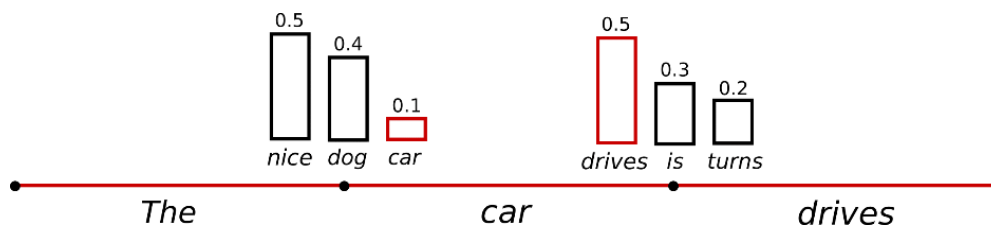


Figure 3.10: Example of sampling for text generation with prior  $x = [The]$  [134].

A simple trick called *temperature* can be applied to control the confidence of the LM. It consists of dividing the result of the model output layer before the softmax activation by a parameter  $t > 0$ . Lower temperatures  $t < 1$  makes the model increasingly confident in its top choices, while  $t > 1$  decreases confidence. Figure 3.11 shows the previous sampling 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 first sampled

word is *nice* followed by the word *house*.

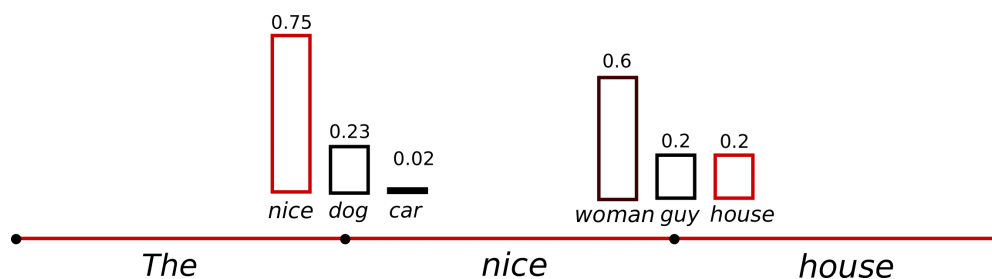


Figure 3.11: Example of sampling for text generation with temperature  $t = 0.7$  and prior  $x = [The]$  [134].

*Top-k sampling* [37] is another way to control the probability distribution of the LM. It consists of using only the  $k$  most likely tokens in the distribution, redistributing the probability mass among only those *top-k* words. With this approach, the LM is filtered at each generation step and the token is picked randomly according to the resulting probability distribution. Figure 3.12 illustrates a text generation example with top-k sampling where  $k = 6$ .

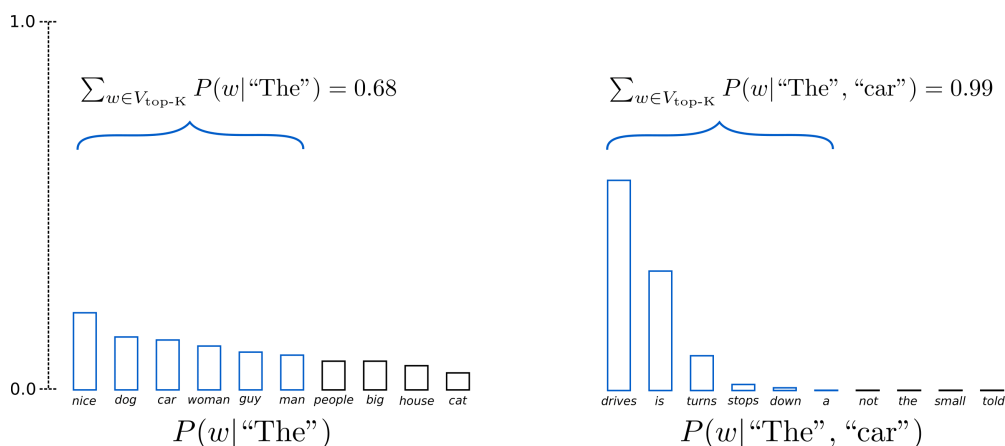


Figure 3.12: Example top-k sampling for text generation with  $k = 6$  and prior  $x = [The]$  [134].

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 selected words  $\{down, a\}$  are arguably bad candidates. Nevertheless, the eliminated words  $\{not, the, small, told\}$  are rather bad candidates and successfully eliminated. This example highlights that top-k sampling can jeopardize the LM, making it to produce wrong sentences for sharp distributions. It can also limit the model’s creativity for flat distributions.

### 3.3.2 Top-p (Nucleus) Sampling

Holtzman et. al. [57] 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 same previous text generation example with top-p sampling with  $p = 0.92$ .

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 words when the next word seems more predictable (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 words while allowing for some dynamic selection.

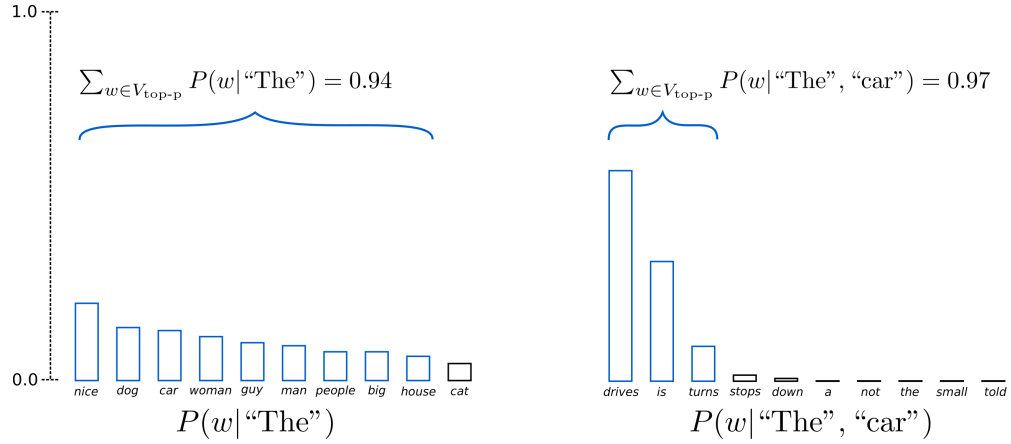


Figure 3.13: Example top-p sampling for text generation with prior  $x = [\textit{The}]$  [134].

### 3.3.3 Greedy Search

Greedy search is the most basic decoding algorithm based on search. It consists of selecting the token with the highest probability at each generation step  $t$ :  $x_t = \textit{argmax} P(x_t | x_{1:t-1})$ . Figure 3.14 shows a text generation example similar to the previous ones but with greedy decoding. Starting with the prior sequence  $x = [\textit{The}]$ , in step  $t = 1$ , the algorithm selects the word *nice* with has the highest probability among the three options  $\{\textit{dog}, \textit{nice}, \textit{car}\}$ . In step  $t = 2$ , the options are  $\{\textit{woman}, \textit{house}, \textit{guy}\}$  and the algorithm greedily selects *woman*. Thus, 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, the greedy search selects *nice* and completely disregards the branch with the word *dog* in step  $t = 2$ .

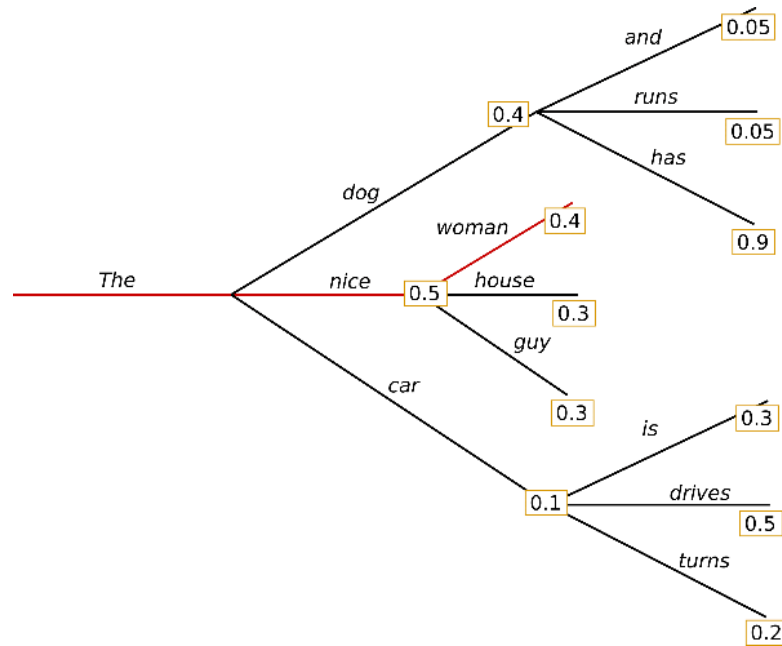


Figure 3.14: Example of greedy search for text generation with prior  $x = [The]$  [134].

### 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 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 sequence.

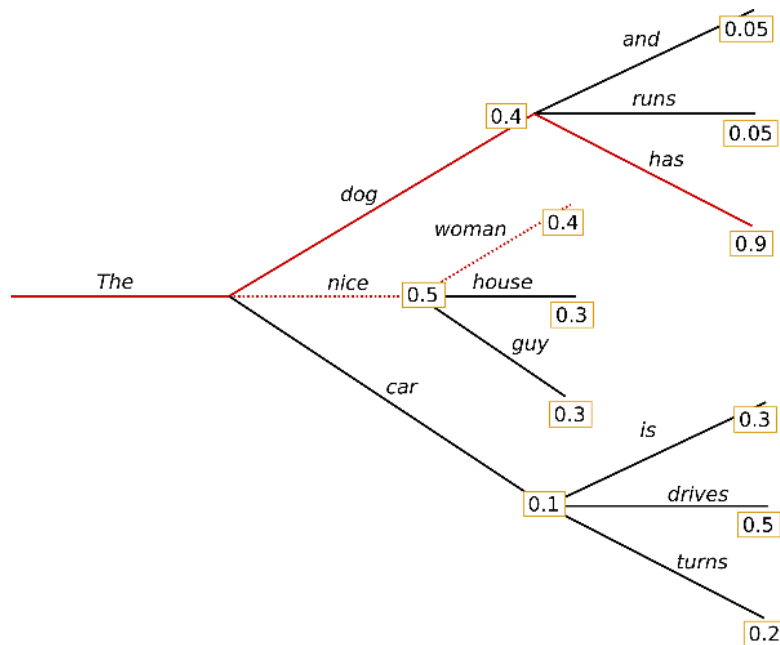


Figure 3.15: Example of beam search for text generation with prior  $x = [The]$  and beam size  $b = 2$  [134].

A common problem of beam search decoding (and other search-based approaches) is that the decoded sequences present became repetitive in a few generation steps [57]. This problem can be alleviated by combining beam search with sampling (including top-k or top-p strategies) to create a *Stochastic Beam Search* [108]. In this case, the sub-sequences are sampled at each time step according to their joint probability, instead of selected greedily.

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, GANs) trained with different *symbolic music* datasets (e.g. MAESTRO, Lakh, 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 perceived emotion of music composed by deep generative models. The next chapter discusses the previous related works with a similar goal of this dissertation: AMC systems that control perceived emotion, controllable LMs, and conditional generative models that control structural music features.



## Chapter 4

# Controllable Algorithmic Music Composition

Deep generative models are currently the leading methods for AMC [144], however, one of the major problems of these methods is to control 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 scene of a movie. Controlling emotions in AMC has been actively investigated in the field of *affective algorithmic composition* [137]. 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 AMC methods that compose music to make a listener perceive or feel a given target emotion [137],

with applications ranging from soundtrack generation [136] to sonification [20] and music therapy [91]. The AAC literature examined various methods, such as expert systems, evolutionary algorithms, Markov chains, and neural networks. 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 qualitative analysis of generated samples or listening tests with human subjects. 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 methodology 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 [33]. This section does not aim to discuss all of them but to provide a definition of emotion helpful for designing neural generative models that can compose music with controllable emotion. According to Williams et al. [137], 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. [137] 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* [32]. Categorical models use discrete labels to classify emotions. For example, Ekman's model [34] divides human emotions into six basic categories: anger, disgust, fear,

happiness, sadness, and surprise. This model builds on the assumption that an independent neural system subserves every discrete basic emotion [32]. Therefore, any other secondary emotion (e.g. rage, frustration, grief) can be derived from the basic ones. Parrott [105] proposed a similar model, but deriving a hierarchical structure with six basic emotions: joy, surprise, anger, sadness, and fear. Basic emotions are expanded into secondary emotions (e.g. passion, pleasure, envy), which in turn derive ternary emotions (e.g. compassion, frustration, 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) [147] 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 [118] 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 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* is another two-dimensional model based on valence and arousal [14]. 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.

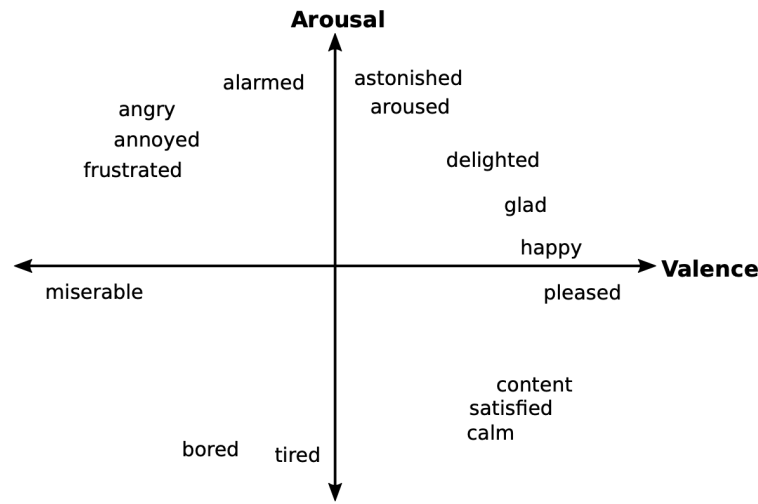


Figure 4.1: The circumplex model of emotion. The horizontal and vertical axes represent valence and arousal, respectively [118].

Independent of the model, emotions can be also classified as *perceived* or *felt*. A person *perceives* an emotion when she objectively recognizes that emotion from her surroundings. For example, one can usually recognize others' emotions using expressed cues, including facial expression, tone of voice, and gestures. The same can happen when one listens to music, in that she recognizes the music as happy or sad using cues such as key, tempo, or volume. A person *feels* an emotion when she actually experiences that emotion herself. 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. Zentner, Grandjean, and Scherer [147] showed that emotions are less frequently felt in response to music than they are perceived as expressive properties of the music.

### 4.1.2 Expert systems

Expert systems are one of the most common methods of AAC [137]. They encode knowledge from music composers to map musical features into a given emotion in a categorical or dimensional space. For example, Williams et al. [136] proposed a system to generate soundtracks for video games from a scene graph<sup>1</sup> annotated according to twelve categorical emotions derived from the circumplex model. A second-order Markov chain was used to learn melodies from a symbolic music dataset which are then transformed by an expert system to fit the annotated emotions in the graph. Each of the twelve emotions was mapped to a different configuration of five music parameters: rhythmic density, tempo, modality (major or minor), articulation, mean pitch range, and mean spectral range. These pre-defined configurations were used to transform the generated melodies. 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. [136] evaluated their system by qualitatively examining a few examples of generated pieces.

TransProse [23] 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 qualitatively analysis of nine music pieces generated by the system for nine respective novels.

---

<sup>1</sup>A graph defining all the possible branching of scenes in the game.

Scirea et al. [120] 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 using a genetic algorithm and (iii) producing an accompaniment, adding 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. [120] 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*.

### **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 perceived emotion. It is very challenging 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. The benefit of this approach 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. However, every time one wants to generate a new set of pieces, the slow interactive evolutionary process has to be restarted.

Kim and André [71] proposed a genetic algorithm to generate polyrhythms<sup>2</sup> for four percussion instruments. It starts with a random population of polyrhythms and evolves them towards a given target emotion (e.g. relaxing or disquieting). 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 (non-pitched) note is played in that division and 0 means silence. The fitness of a polyrhythm is given by a human subject who listens to it and 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. [149] presented an interactive genetic algorithm based on the KTH rule system [41]. This system models performance principles within the realm of Western classical, jazz and popular music. These rules control different music performance parameters (e.g. phrasing, articulation, tonal tension) with weights called *k values* that control the magnitude of each rule. Zhu et al. [149] encoded the individuals as a set of *k values* used to create MIDI performances of the given 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.

---

<sup>2</sup>A polyrhythm is the concurrent playing of two or more different rhythms.

Nomura and Fukumoto [97] designed a distributed interactive genetic algorithm 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 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.

#### **4.1.4 Markov Chains**

AAC systems based on Markov chains are typically hybrid systems where Markov chains compose an underlying piece of music which is transformed by an expert system. The work of Williams et al. [136] described earlier in this chapter is an example of such a system. Ramanto and Maulidevi [113] proposed a similar system in which the Markov chain is designed manually by the authors, instead of derived from a corpus. There are not a lot works training Markov chains directly from datasets of symbolic music labelled according to a model of emotion. One of the major examples is the work of Monteith et al. [93, 19], which use different Markov models to generate polyphonic music from a MIDI corpus labelled according to the Parrott model of emotion [105]. The corpus is composed by 45 movie soundtracks labelled by



6 researchers using the 6 basic Parrott emotions: 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 accompaniment/instrumentation planner. First, the rhythm generator randomly selects a phrase from the subset of pieces with the target emotion and transforms its rhythm by randomly changing a number of measures. Second, the pitch generator assigns pitch values to the notes in the rhythm by sampling from a n-gram (i.e. 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 in which pitches are considered observed events and the chord progression underlying state sequence. Finally, the instrumentation planner probabilistically selects the instruments for melody 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 (positive) or not (negative). The system only accepts rhythms and melodies that are classified as positive by the neural networks. Monteith et al. [93] evaluated their system with a listening test in which thirteen human subjects listened to twelve generated pieces. Each subject judged a piece by selecting the emotion that they perceived in the piece. Moreover, the subject used two 10-point Likert scales to evaluate how human-like, and unique the pieces are.

#### **4.1.5 Deep Learning**

Deep NNs started being used for AAC very recently. Indeed, this dissertation is one of the first works in this area. One of the main challenges of using deep NNs for AAC is the lack

of labelled data. It is considerably expensive to create a large dataset of symbolic music labelled according to a model of emotion. However, a few works have been proposed. Madhok et al. [88] introduced one of the few works prior to this dissertation with the framework SentiMozart, which generates piano pieces that match the emotion of a given facial expression. SentiMozart 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 [47] dataset, which has 35,887 images of facial expressions labelled with the following emotions: angry, disgust, fear, happy, sad, surprise, and neutral. To train the LSTM, Madhok et al. [88] created a dataset with 200 MIDI piano pieces, each labelled by 15 annotators according to the classes happy, sad, and neutral. An LSTM was trained independently for each of these three emotions. To unify the models of emotion between the two datasets, Madhok et al. [88] 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 of the convolutional NN was one of these three emotions and was used to select the respective LSTM that, in turn, composes the music with the given emotion. SentiMozart was evaluated with a listening test where thirty human subjects judged 30 randomly chosen images (10 of each class) and their corresponding generated music. The participants used a ten-point Likert scale in which 0 means sad, 5 means neutral, and 10 means happy.

Tan and Antony [129] proposed a deep generative NN for a similar problem: generating music that matches an emotion expressed by visual artworks (e.g. paintings, illustrations, 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 [51]. Tan and Antony [129] compared two decoders: a LSTM and a transformer decoder. The music pieces were encoded as sequences of tokens with the MIDI-based method proposed by Oore et al. [100]. Tan and Antony [129] trained their model by pairing emotion-labeled images from You et al. [145] (17,349 images) with emotion-labeled music pieces from the multi-modal MIREX-like emotion dataset (196 MIDI files) [103]. Since these two datasets use different categorical models of emotion, Tan and Antony [129] 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. Six human subjects performed the listening test. They were asked to evaluate the sentiment of the music pieces and the image on a ten-point Likert scale without knowing which piece of music 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. [148] presented a conditional Biaxial LSTM [66] to generate piano pieces with controllable emotion. They labelled the *Piano midi.de* dataset according to a categorical model of emotion: happy, sad, peaceful, and tensional. They used a piano-roll representation for the input music pieces, which are processed by their model 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. [148] evaluated the quality of the generated pieces with 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 [128] is a framework based on a Gaussian Mixture Variational Au-

toencoder (GM-VAE) that can learn high-level music feature representations (such as emotion) by modelling corresponding low-level structural music features. Based on the work Yang et al. [144], Music FaderNets learns 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. 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 sequence of tokens [100] extracted from the Yamaha Piano-e-Competition dataset [1] and the VGMIDI dataset<sup>3</sup> [40]. Results showed that Music Fadernets successfully learns the intrinsic relationship between arousal and its corresponding low-level attributes rhythm and note density.

## 4.2 Controllable Neural Language Models

Considering the similarities between music and text neural LMs, this dissertation is also related to methods that control neural LMs to generate text with given characteristics (e.g. topic, style, sentiment). For example, Radford et al. [109] showed that an LSTM can learn an excellent representation of sentiment (positive-negative) on text, despite being trained only to predict the next character in the Amazon reviews dataset [52]. When fine-tuned with an extra linear layer, this LSTM achieved state-of-the-art sentiment analysis on the stanford sentiment treebank dataset [124] and matched the performance of previous supervised systems using 30-100x fewer labeled examples. Radford et al. [109] observed that this LSTM stored almost all of the sentiment signal in a distinct *sentiment neuron*, which could be used to control the LSTM to generate sentences with a given sentiment.

---

<sup>3</sup>The VGMIDI dataset is contribution of this dissertation described in Chapter 5

CTRL [70] is a transformer LM trained to generate text conditioned on special tokens that inform the LM about the characteristics of the text to be generated (e.g., style). These control codes are derived automatically from the origin of the text (e.g., Wikipedia), which means no manual annotation had to be performed. During training, every example is fed as input together with its control codes. CTRL can be used to generate text with a particular style, for example, by feeding the information that represents that style. This dissertation differs from CTRL because we control the LM with a search procedure and not with an extra input to the LM. Conditioning the LM requires a large amount of labeled data, which is expensive in the affective music domain.

Holtzman et al. [56] proposed a decoding strategy with a set of discriminator NNs that steer the probability distribution of a LM during generation time. At each decoding step, the probabilities of the next word 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  $(x, y_g)$ , where  $x$  is a prior context sentence, and  $y_g$  is the completion of that sentence. At each independent training phase, the loss function measures the difference between the scores assigned to the truth continuation  $y_g$  and the scores assigned to the continuation generated by the LM  $\hat{y}$ . Once all the discriminators are trained, Holtzman et al. [56] optimize the weight coefficients that will be used when decoding the LM.

The Plug and Play LM [22] combines a pre-trained LM  $L$  with an attribute (e.g. sentiment) classifier  $C$  to guide text generation by fine-tuning the hidden layers of the LM in decoding time. At each generative time-step, the hidden layers of LM 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  $L$ . The shifting process occurs in three steps: (1) a first 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) a final forward pass to update the distribution over the vocabulary from the updated LM hidden layers. With this approach, multiple attribute classifiers can be combined during the generation with customized weights.

Given a music dataset labeled according to emotion, one could apply the NLP methods discussed in this section to control music LMs to generate pieces with a target emotion. Indeed, the first two contributions of this dissertation are inspired by such methods. The next chapter presents a method based on Radford et al. [109] to control the sentiment of piano pieces generated by an LSTM. Chapter 6 discusses a variation of beam-search inspired by Holtzman et al. [56] to generate music with controllable emotion for tabletop role-playing games.

## Chapter 5

# Learning to Generate Music with Sentiment

### 5.1 Introduction

This chapter introduces a method inspired by Radford et al. [109] to generate symbolic music with sentiment. Radford et al. [109] fine-tuned an LSTM LM with an extra linear layer to classify sentiment in text. This fine-tuning procedure highlighted that most of the sentiment signal was stored in a single neuron. Thus, this neuron could be directly adjusted to control the original LSTM LM to generate text with a target sentiment. This chapter shows an application of Radford et al. [109]’s method to control sentiment in symbolic music generated via LMs. First, the VGMIDI dataset is introduced, a new dataset of symbolic piano pieces labeled according to a circumplex model of emotion. Second, an LSTM is trained as a LM with unlabeled piano pieces. Third, this LSTM-based LM is fine-tuned with an extra linear layer on the valence (i.e. sentiment) dimension of the VGMIDI dataset. Different than the findings of Radford et al. [109], the fine-tuning step didn’t store the sentimental in a single neuron but in many neurons

in a more balanced way. Thus, a genetic algorithm is applied to optimize the weights of these sentiment neurons towards a given sentiment. The accuracy of the model in classifying sentiment of symbolic music is evaluated with the VGMIDI dataset, and results show that the model is able to obtain good prediction accuracy. Moreover, a user study shows that human subjects agreed that the generated music has the intended sentiment; however, negative pieces can be ambiguous. This work has been published at the 20th annual conference of the International Society for Music Information Retrieval (ISMIR).

## **5.2 The VGMIDI dataset**

Currently, there are only a few datasets of symbolic music labelled according to emotion [88, 129, 148] and none are labelled according to sentiment. Thus, a new dataset called VGMIDI has been created with 823 pieces extracted from video game soundtracks in MIDI format. Video game soundtracks are well suited for this dataset because they are typically composed to intentionally keep the player in a specific affective state and thus tend to be less subjective. All the pieces are piano arrangements of the soundtracks and they vary in length from 26 seconds to 3 minutes. Among these pieces, 95 are annotated according to a circumplex model that represents emotion using a valence-arousal pair. VGMIDI uses a valence-arousal 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 (happy, sad, surprise, etc) or a binary (positive/negative) model. Thus, the same set of labelled data permits the investigation of affective algorithmic music composition as both a classification (multiclass and/or binary) and as



a regression problem. The valence-arousal model is also one of the most common dimensional models used to label emotion in music [125].

Annotating a piece according to the circumplex model consists of continuously listening to the piece and deciding what valence-arousal 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 video game soundtracks in MIDI format is composed of five steps, each one being a single web page. These steps are based on the methodology proposed by Soleymani et al. [125] for annotating music pieces in audio waveform. 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 valence-arousal 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 of the pieces the annotators listened to are MIDI files synthesized with the “Yamaha C5 Grand” soundfont. 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% classified themselves as transgender female, transgender male, genderqueer, or choose not to disclose their gender. All annotators are from the United States

## Music Emotion Annotation Task

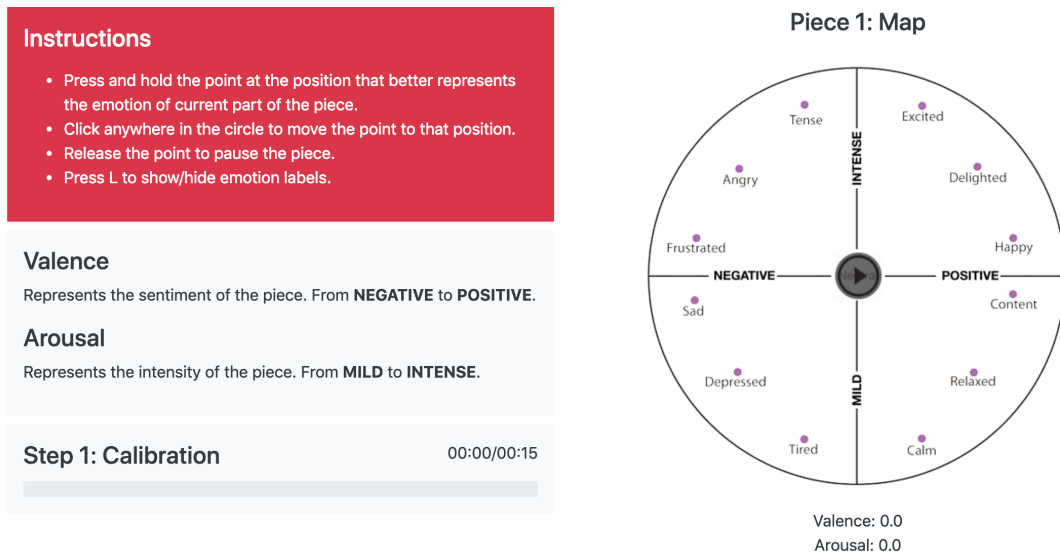


Figure 5.1: Screenshot of the annotation tool.

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. They spent on average 12 minutes and 6 seconds to annotate the two pieces.

The data collection process provides a time series of valence-arousal 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 preprocessed, summarized into one time series, and split into “phrases” of the same sentiment. The preprocessing is intended to remove noise caused by subjects performing the task randomly to get the reward as fast as possible. The data was preprocessed by smoothing each annotation

with a moving average and clustering all 30 time series into 3 clusters (positive, negative, and noise) according to the dynamic time-warping distance metric.

The cluster with the highest variance is considered noise, and so it is discarded. The cluster with more time series among the two remaining ones is then selected and summarized by the mean of its time series. The mean is split at all the points where the valence changes from positive to negative or vice-versa. This process creates several segments with the same sentiment, where segments with negative valence are considered negative phrases and segments with positive valence are positive phrases. Figure 5.2 shows an example of this three-step process performed on a piece. 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 Models and Data Representation**

Radford et al. [109] used a single-layer multiplicative LSTM (mLSTM) network [77] 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

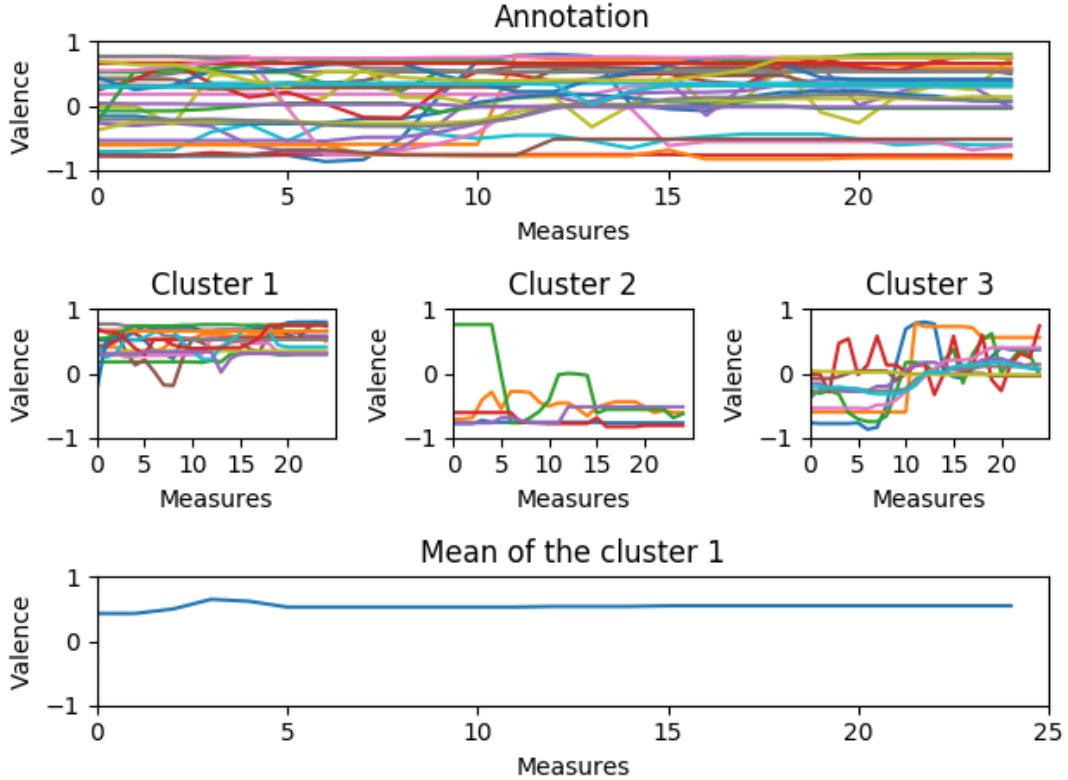


Figure 5.2: Data analysis process used to define the final label of the phrases of a piece.

to several features such as melody, harmony, tempo, and timbre[72]. This data representation attempts to encode a large part of these features<sup>1</sup> using a small set of words:

- “n\_[pitch]”: play note with given pitch number: any integer from 0 to 127.
- “d\_[duration]\_[dots]”: change the duration of the following notes to a given duration type with a given amount of dots. Types are breve, whole, half, quarter, eighth, 16th and 32nd.

Dots can be any integer from 0 to 3.

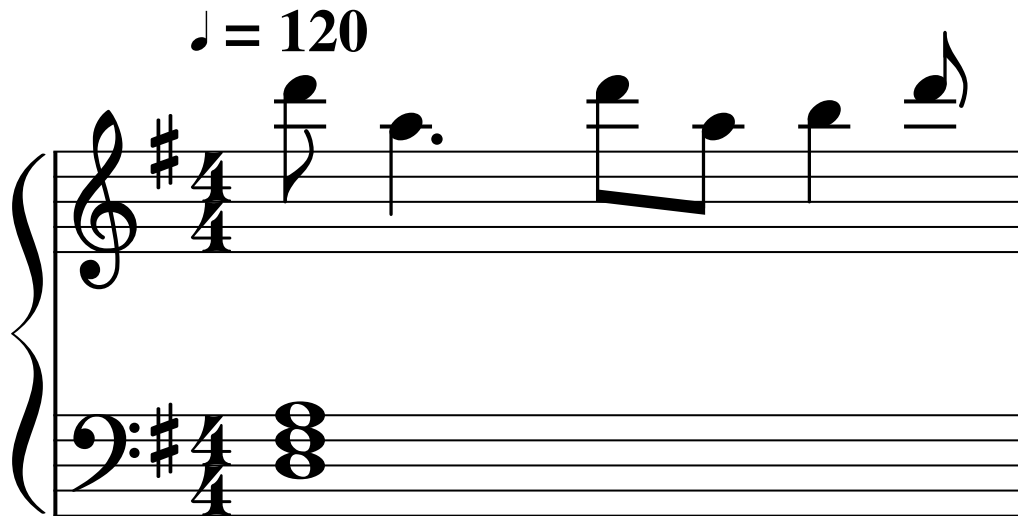
<sup>1</sup>Constrained by the features one can extract from MIDI data.

- “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  $V = 4, 8, 12, \dots, 128$ .
- “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  $T = 24, 28, 32, \dots, 160$ .
- “.”: end of time step. Each time step is one sixteenth note long.
- “\n”: end of piece.

For example, Figure 5.3 shows the encoding of the first two time steps of the first measure of the Legend of Zelda - Ocarina of Time’s Prelude of Light. 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).

## 5.4 Sentiment Analysis Evaluation

The fine-tuning approach (LM + linear sentiment classification layer) proposed in this work is initially evaluated in the sentiment classification task. First, a mLSTM LM is pre-trained with the unlabelled pieces of the VGMIDI dataset. Then, the pre-trained LM layers are frozen, and an additional linear layer is stacked on top of it. The resulting model is trained as a sentiment classifier with the labeled pieces of the VGMIDI dataset. This fine-tuned LM model



```
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.

was compared against a baseline mLSTM network trained directly on the supervised sentiment classification task. The baseline method uses only labeled MIDI phrases to train a classification mLSTM to predict the sentiment for the phrase.

The unlabelled pieces used to train the LM were augmented in order to create additional training examples, following the methodology of Oore et al. [100]. 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 ??). Finally, the encoded pieces were shuffled, and 90% of them were used for training and 10% for testing. The training set was divided into three shards of similar size (approximately 18,500

pieces each – 325MB), and the testing set was combined into one shard (approximately 5800 pieces – 95MB).

Six different sizes (number of neurons in the mLSTM layer) were compared for the LM: 128, 256, 512, 1024, 2048, and 4096. For each size, the LM was trained for 4 epochs using the 3 training shards. Weights were updated with the Adam optimizer after processing sequences of 256 words on mini-batches of size 32. The mLSTM hidden and cell states were initialized to zero at the beginning of each shard. They were also persisted across updates to simulate full-backpropagation and allow for the forward propagation of information outside of a given sequence [109]. 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 was set to  $5 * 10^6$  at the beginning and decayed linearly (after each epoch) to zero over the course of training.

Each variation of the LM was evaluated with a forward pass on the test shard using mini-batches of size 32. Table 5.1 shows the average<sup>2</sup> cross entropy loss for each variation of the mLSTM LM. The average cross entropy loss decreases as the size of the mLSTM increases, reaching the best result (loss 1.11) when size is equal to 4096. Thus, the variation with 4096 neurons was used to proceed with the sentiment classification experiments.

After training the LM, an extra linear sentiment classification layer is stacked on top of the base LM and trained with the 966 labeled phrases of the VGMIDI dataset. Here, stacking means passing the final cell states of the mLSTM as input to the linear classification layer. During this fine-tuning step, the base layers of the LM are frozen, which means only the

---

<sup>2</sup>Each mini-batch reports one loss.



<b>mLSTM Neurons</b>	<b>Average Cross Entropy Loss</b>
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 generative mLSTM with different amount of neurons.

weights of the stacked classification layer are updated. Moreover, these extra layer is trained with L1 regularization to shrink the least important of the 4096 feature weights to zero. This ends up highlighting the generative mLSTM neurons that contain most of the sentiment signal.

The fine-tuning approach (LM + linear sentiment classification layer) was compared against the baseline supervised mLSTM. This is an mLSTM with exactly the same architecture and size as the fine-tuned version but trained in a fully supervised way. This supervised mLSTM was trained with the same word-based representation of the phrases, but they were padded with silence (the symbol “.”) in order to equalize their length. Training parameters were set to be the same used to train the base LM. 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.

<b>Method</b>	<b>Test Accuracy</b>
Fine-tuned mLSTM-4096	$89.83 \pm 3.14$
Baseline mLSTM-4096	$60.35 \pm 3.52$

Table 5.2: Average (10-fold cross validation) sentiment classification accuracy of both generative (with logistic regression) and supervised mLSTMs.

The fine-tuned mLSTM achieved an accuracy of 89.83%, outperforming the super-

vised mLSTM by 29.48%. The supervised mLSTM accuracy of 60.35% suggests that the amount of labeled data (966 phrases) was not enough to learn a good mapping between phrases and sentiment. The accuracy of the fine-tuned mLSTM shows that it 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 fine-tuned mLSTM is derived from unlabeled data, it will be easier to improve this over time using additional (less expensive) unlabeled data instead of the supervised mLSTM approach, which requires additional (expensive) labeled data. Second, because the mLSTM LM was trained to predict the next word in a sequence, it can be used as a music generator. Since it 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.

## 5.5 Generative Evaluation

To control the sentiment of the music generated by the fine-tuned mLSTM, one has to find the subset of neurons that contain the sentiment signal by exploring the weights of the linear sentiment classification layer. As shown in Figure 5.4, the linear classification layer trained with L1 regularization uses 161 neurons out of 4096. Unlike the results of Radford et al. [109], the fine-tuning step did not store the sentiment 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 neurons' weights to

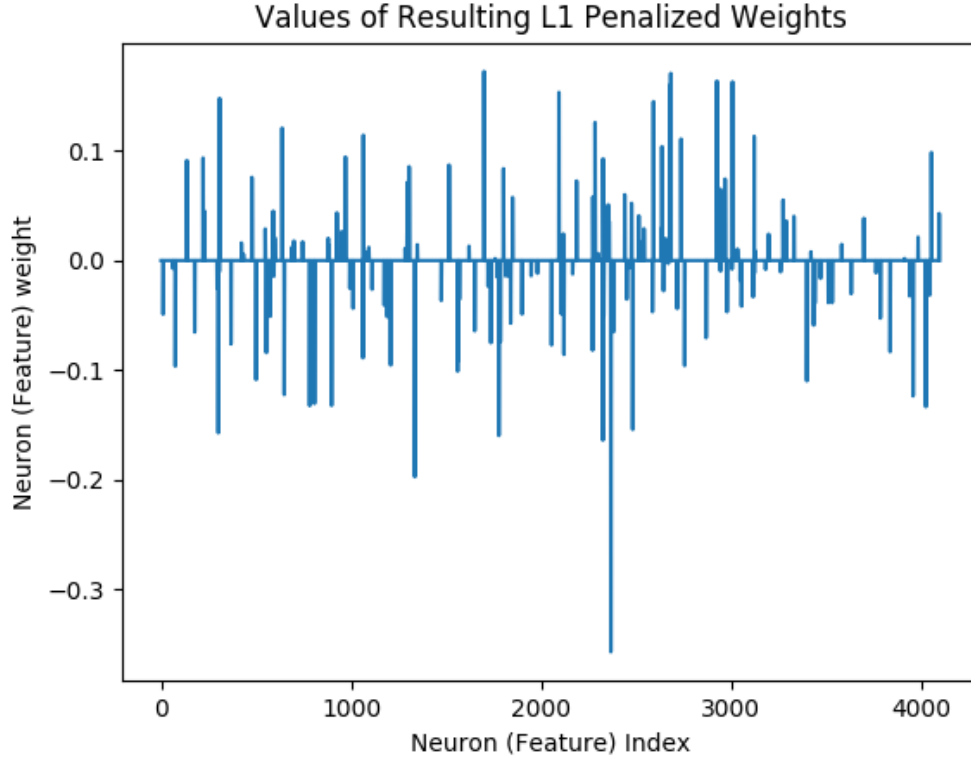


Figure 5.4: Weights of 161 L1 neurons. Note multiple prominent positive and negative neurons.

lead the fine-tuned mLSTM to generate only positive or negative pieces. Each individual in the population of this GA has 161 real-valued genes representing a small noise to be added to the weights of the 161 L1 neurons. The fitness of an individual is computed by (i) adding the genes of the individual to the weights (vector addition) of the 161 L1 neurons of the generative mLSTM, (ii) generating  $P$  pieces with this mLSTM, (iii) using the logistic regression model 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 indi-

vidual 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 the fine-tuned mLSTM for generating positive pieces and once for negative pieces. Each execution optimized the individuals during 100 epochs with a crossover rate of 95% and 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 genes of the best individual of the final population to the weights of the generative mLSTM, 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 weights of the 161 L1 neurons of the trained LM. 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 neg-

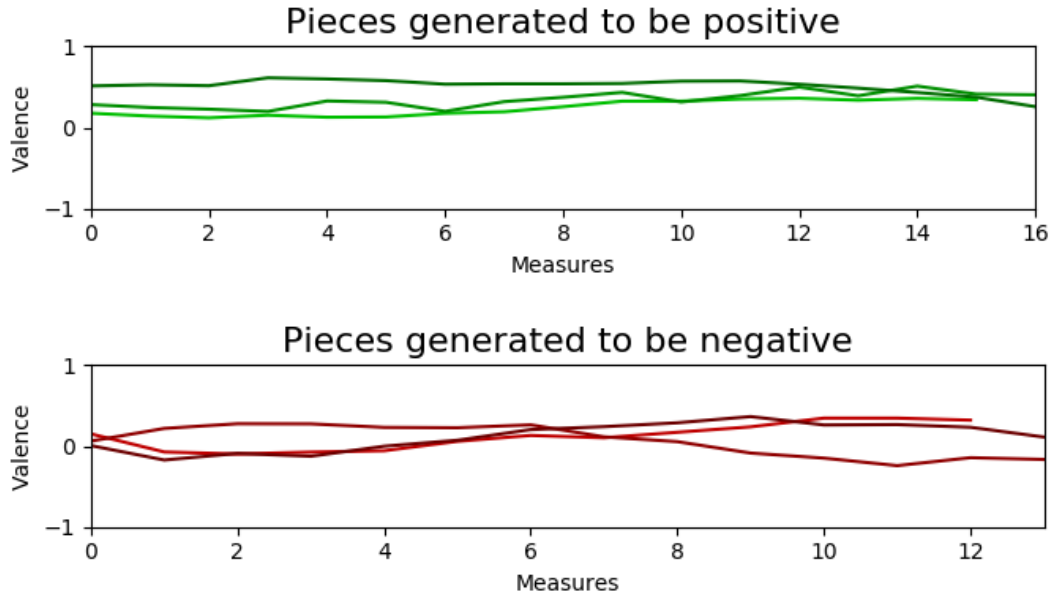


Figure 5.5: Average valence of the 6 generated pieces, as determined by human annotators, with least variance.

ative 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 mLSTM to generate complete negative pieces. Moreover, the challenge to optimize the L1 neurons suggests that there are more positive pieces than negative ones in the 3 shards used to train the generative mLSTM.

## 5.6 Conclusions

This chapter presented a mLSTM LM that can be controlled 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 fine-tuning approach 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 mLSTM 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 *Bardo Composer*, or *Composer* for short, a system to generate background music for tabletop role-playing games (TRPG). *Composer* is applied to score sessions of Dungeons and Dragons (D&D), a TRPG where players interpret characters of a story conducted by a special player called the dungeon master. For example, if the players are fighting a dragon, *Composer* should generate a piece matching such an epic moment of the story. TRPG players often manually choose songs to play as background music to enhance their experience [10]. Composers' goal is to augment the players' experience with soundtracks that match the story being told in the game. Importantly, 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*, presented by Padovani et al. [101] to select pre-authored background music for tabletop games. Bardo uses a *naive bayes* approach to classify 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 player speech into text, which is classified by a transformer NN according to a categorical model of emotion. A second transformer NN is trained to classify the emotion of symbolic piano pieces. Bardo Composer then uses Stochastic Bi-Objective Beam Search, a new decoding strategy inspired by Holtzman et al. [56], to generate musical pieces conveying the desired emotion. Stochastic Bi-Objective Beam Search works by steering the probabilities given by the LM with the probabilities given by the music emotion classifier.

Composer uses a discretized circumplex emotion model [118] that generalizes the model used in Bardo. In Composer’s emotion model, the dimensions of valence and arousal, denoted by a pair  $(v, a)$ , assume binary values  $v \in [0, 1]$  and  $a \in [0, 1]$ , respectively. Valence measures sentiment and thus  $v = 0$  means a negative input and  $v = 1$  means a positive input. Arousal measures the energy of the input, and thus  $a = 0$  means that the input has low energy, whereas  $a = 1$  means that the input has high energy. This model is used for classifying both



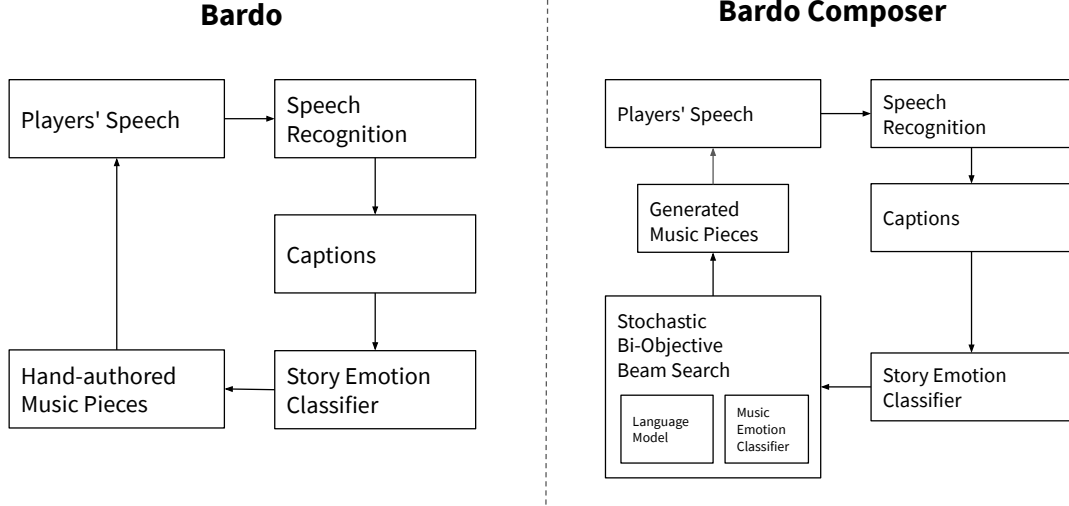


Figure 6.1: Diagrams with the architecture of Bardo (left) and Composer (right).

the emotion of the player’s speeches and the emotion of the generated music. Composer uses Padovani et al. [101]’s dataset to train an emotion classifier for the story being told at a game session. Their dataset includes 9 episodes of Call of the Wild (CotW), a Dungeons and Dragons campaign available on YouTube. This dataset contains 5,892 sentences and 45,247 words, resulting in 4 hours, 39 minutes, and 24 seconds of gameplay. There are 2,005 Agitated, 2,493 Suspenseful, 38 Happy, and 1,356 Calm sentences in the dataset.

A user study with 116 participants was performed to evaluate whether people are able to correctly identify the emotion conveyed in pieces generated by Composer for the CotW campaign. Results showed that human subjects could correctly identify the emotion of the generated music pieces as accurately as they were able to identify the emotion of pieces written by humans. This work has been published at the 16th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE).

## 6.2 Bardo Composer

A general overview of Composer is shown in Algorithm 2. It receives as input a speech recognition system  $S$ , an emotion classifier for text  $E_s$ , an emotion classifier for music  $E_m$ , a LM for symbolic music generation  $L$ , a speech signal  $v$  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 the search procedure described in Algorithm 3. Composer returns a symbolic piece that tries to match the emotion in the players' speeches. Composer starts by converting the speech signal  $v$  into text  $s$  with  $S$  (line 1). In addition to text,  $S$  returns the duration of the signal  $v$  in seconds, this is stored in  $l$ . Then, Composer classifies the emotion of  $s$  in terms of valence  $v$  and arousal  $a$  and it invokes Stochastic Bi-Objective Beam Search (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  $k$ , which are explained below, 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  $v$ , previously composed symbols  $x$ , beam size  $b$ , number of symbols  $k$

**Ensure:** Music piece  $x$

- 1:  $s, l \leftarrow S(v)$
  - 2:  $v, a \leftarrow E_s(s)$
  - 3:  $y \leftarrow \text{SBBS}(L, E_m, x, v, a, b, k, l)$  # 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  $v$  and returns a composed piece of music. While the most recent piece is being played at the game table, Composer receives another signal  $v$  and composes the next excerpt. One also needs to define the length of the signal  $v$ . Similar to Bardo Padovani et al. [101], Composer uses YouTube’s subtitle system as the speech recognition system  $S$ . Therefore, signals  $v$  are long enough to form a sentence in the form of a subtitle.

### 6.2.1 Classifying the Story’s Emotion

In order to have a common model of emotion between stories and music, Bardo’s four emotion model is mapped to the circumplex model used by Composer:

1. Suspenseful is mapped to low valence and arousal ( $v = 0, a = 0$ ).
2. Agitated is mapped to low valence and high arousal ( $v = 0, a = 1$ ).
3. Calm is mapped to high valence and low arousal ( $v = 1, a = 0$ ).
4. Happy is mapped to high valence and arousal ( $v = 1, a = 1$ ).

The valence-arousal mapping is based on the model used to annotate the VGMIDI dataset. When human subjects annotated that dataset, they used a continuous valence/arousal

model with labels defining a fixed set of discrete basic emotions (e.g. happy or sad) [40]. This mapping allows one to use the circumplex model with the labeled CotW dataset. For example, in the context of the game Dungeons and Dragons, the sentence “Roll initiative” is normally said at the beginning of battles and it can be considered ( $v = 0, a = 1$ ), once a battle is a negative (dangerous) moment with high energy. “Roll initiative” is normally classified as Agitated in Padovani et al. [101]’s dataset.

Given the limited amount of TRPG stories labeled according to emotion (there are only 5,892 sentences in the CotW dataset), the sentences [110] are classified with a transfer learning approach. A high capacity BERT [24] model is fine-tuned with the CotW dataset. BERT is used because it is currently one of the state-of-the-art architectures across different NLP tasks [24]. Although in Algorithm 2 the classifier for story emotions is depicted as a single  $E_s$  model, valence and arousal are treated independently in practice, thus a pre-trained BERT is fine-tuned for each dimension.

### 6.2.2 Classifying the Music’s Emotion

As was the case with the TRPG stories, given the limited amount of MIDI pieces labeled according to emotion, the emotion music classifier ( $E_m$ ) is also trained with a transfer learning approach. However, different than the  $E_s$ , where a BERT model is fine-tuned,  $E_m$  uses a fine-tuned GPT-2 model.  $E_m$  uses a GPT-2 because it is better suited for sequence generation than BERT. Similar to  $E_s$ ,  $E_m$  also treats valence and arousal independently. Thus, a pre-trained GPT-2 is fine-tuned for each of these dimensions.

Currently, in the symbolic music domain, there is no publicly available high-capacity

LM pre-trained with large (general) datasets. Typically, models in this domain are trained with relatively small and specific datasets. For example, the MAESTRO dataset [50], the Bach Chorales [49] and the VGMIDI [40] dataset. Therefore, a general high-capacity GPT-2 has been pre-trained as a LM [111] using a new dataset called ADL (Augmented Design Lab) Piano MIDI dataset <sup>1</sup>. The ADL Piano MIDI dataset is based on the Lakh MIDI dataset [112], which is one of the largest MIDI datasets publicly available. Among these files, there are many versions of the same piece. Only one version of each piece was kept. Given that the VGMIDI dataset is limited to 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 generated a total of 9,021 unique piano MIDI files. These files are mainly Rock and Classical pieces, so to increase the genre diversity (e.g. Jazz, Blues, and Latin) of the dataset, an additional 2,065 files were included from public sources on the Internet<sup>2</sup>. All files in the final collection were de-duped according to their MD5 checksum. The final dataset has 11,086 pieces.

After pre-training the high-capacity GPT-2 model, two independent models were fine-tuned (one for valence and one for arousal) with an extended version of the VGMIDI dataset[40]. Following the approach of Radford et al. [110], these models were fine-tuned by adding an extra layer to the pre-trained LM and training the entire model (including the pre-trained layers) with the VGMIDI dataset [40]. In this work, the VGMIDI dataset has been extended from 95 to 200 labeled pieces using the same annotation method of the original dataset. All the 200 pieces are piano arrangements of video game soundtracks labeled according to the

---

<sup>1</sup><https://github.com/lucasnfe/adl-piano-midi>

<sup>2</sup><https://bushgrafts.com/midi/> and <http://midkar.com/jazz/>

circumplex model of emotion.

### 6.2.2.1 Encoding

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 correspondent `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  $x = \{z_v^1, z_d^1, z_p^1, \dots, z_v^n, z_d^n, z_p^n\}$  of tokens for a given MIDI file by (a) parsing all notes  $z^i$  from the file, (b) sorting them by starting time  $z_s^i$  and (c) concatenating their velocity  $z_v^i$ , duration  $z_d^i$  and pitch  $z_p^i$ . 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  $|V| = 314$ .

### 6.2.3 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 stochasticity of SBBS allows it to generate a large variety of musical pieces for the same values of  $v$  and  $a$ . SBBS is “bi-objective” because it optimizes for realism and emotion. The pseudocode of SBBS is shown in Algorithm 3. The letters  $x, y$  and  $m$  denote sequences of musical symbols. Function  $p_L(y) = \prod_{y_t \in y} P(y_t | y_0, \dots, y_{t-1})$  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  $j$ .

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). 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 largest  $p_L$ -value 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

---

**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.

```
1:  $B \leftarrow [x], j \leftarrow 0$ 
2: while  $l(y[t : t + j]) < l, \forall y \in B$  do
3:    $C \leftarrow \{\}$ 
4:   for all  $m \in B$  do
5:      $C_m \leftarrow \{m \cup s | s \in V\}$ 
6:      $C_m \leftarrow k$  elements  $y$  from  $C_m$  with largest  $p_L(y)$ 
7:      $C \leftarrow C \cup C_i$ 
8:   end for
9:    $B \leftarrow b$  sequences  $y$  sampled from  $C$  proportionally to  $p_L(y)(1 - |v - E_{m,v}(y)|)(1 - |a - E_{m,a}(y)|)$ 
10:   $j \leftarrow j + 1$ 
11: end while
12: return  $m \in B$  such that  $p_L(m) = \max_{y \in B} p_L(y)$  and  $l(y[t : t + j]) \geq l$ 
```

---

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).

## 6.3 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 model for story emotion classification is compared against the simpler Naïve Bayes approach of Padovani et al. [101]. The fine-tuned GPT-2 model for music emotion classification is compared against the simpler LSTM of [40]. The second experiment evaluates, with a user study, whether human subjects can recognize different emotions in pieces generated by Composer for the CotW campaign.

### 6.3.1 Emotion Classifiers

#### 6.3.1.1 Story Emotion

Composer’s story emotion classifier is a pair of pre-trained BERT<sub>BASE</sub> models [24], one for valence and one for arousal. BERT<sub>BASE</sub> has 12 layers, 768 units per layer, and 12 attention heads. BERT<sub>BASE</sub> was pre-trained using both the BooksCorpus (800M words) [150] and the English Wikipedia (2,500M words). These two BERT models are independently fine-tuned as valence, and arousal classifiers using the CotW dataset [101]. Fine-tuning consists of adding a classification head on top of the pre-trained model and training all the parameters

Alg.	Episodes									Avg.
	1	2	3	4	5	6	7	8	9	
<b>NB</b>	73	88	91	85	94	81	41	74	94	80
<b>BERT</b>	<b>89</b>	<b>92</b>	<b>96</b>	<b>88</b>	<b>97</b>	<b>81</b>	<b>66</b>	<b>83</b>	<b>96</b>	<b>87</b>

Table 6.1: Valence accuracy in % of Naïve Bayes (NB) and BERT for story emotion classification.

(including the pre-trained ones) of the resulting model end-to-end. All these parameters were fine-tuned for 10 epochs with the Adam optimizer [73]. 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 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 [140] with a 30,000 token vocabulary.

The fine-tuned BERT classifiers are compared with a Naïve Bayes (NB) approach (baseline), chosen because it is the method underlying the original Bardo system. NB encodes sequences using a traditional bag-of-words with tfidf approach. Table 6.1 shows the accuracy of the valence classification of both these methods per episode.

The best accuracy for a given episode is highlighted in bold. The BERT 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 had the best performance improvement when compared to NB. The story told in episode 7 of CotW is different from all other episodes. While the other episodes are full of battles and ability checks, episode 7

is mostly the players talking with non-player characters. Therefore, what is learned in the other episodes does not generalize well to episode 7. The improvement in accuracy of the BERT model in that episode is likely due to the model’s pre-training. Episodes 5 and 9 were equally easy for both methods because they are similar to one another. The system trained in one of these two episodes generalizes well to the other.

Table 6.2 shows the accuracy of arousal classification of both NB and BERT. The best accuracy for a given episode is highlighted in bold. Again BERT 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 the BERT model substantially outperforms NB.

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

Table 6.2: Arousal accuracy in % of Naïve Bayes (NB) and BERT for story emotion classification.

### 6.3.1.2 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  $p$  of this dataset was 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% [100]. Thus, each piece generated  $12 \cdot 3 \cdot 3 = 108$  different examples. The pre-trained GPT-2 LM has 4 layers (transformer blocks), context size of 1024 tokens, 512 embedding units, 1024 hidden

units, and 8 attention heads. The pre-trained GPT-2 LM was then fine-tuned independently using the VGMIDI dataset, for valence and for arousal. Similarly to BERT, fine-tuning a GPT-2 architecture consists of adding an extra classification head on top of the pre-trained model and training all parameters end-to-end. Similar to the story emotion classifiers, the GPT-2 classifiers are fine-tuned for 10 epochs using an Adam optimizer with learning rate  $3e-5$ . Different from the story emotion classifiers, each training step was performed over mini-batches of size 16 (due to GPU memory constraints) and dropout of 0.25. The VGMIDI dataset is defined with a train and test splits of 160 and 40 pieces, respectively. The dataset was augmented by slicing each piece 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 classifier generalize for pieces with different lengths.

The fine-tuned GPT-2 classifiers are compared with LSTM models that were also pre-trained with the ADL Piano Midi dataset and fine-tuned with the VGMIDI dataset. LSTMs were chosen because they are currently the state-of-the-art model in the VGMIDI dataset [40]. The LSTMs have same size as the GPT-2 models (4 hidden layers, 512 embedding units, 1024 hidden units) and were 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) are also reported. These are the baseline versions of the models.

Results show that using transfer learning can substantially boost the performance of both models. The fine-tuned GPT-2 is 10% more accurate in terms of valence and 8% in terms of arousal. The fine-tuned LSTM is 5% more accurate in terms of valence and 12% in terms

Algorithm	Valence	Arousal
Baseline LSTM	69	67
Fine-tuned LSTM	74	79
Baseline GPT-2	70	76
Fine-tuned GPT-2	<b>80</b>	<b>82</b>

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

of arousal. Finally, the fine-tuned GPT-2 outperformed the fine-tuned LSTM by 6% and 3% in terms of valence and arousal, respectively.

### 6.3.2 User Study

Composer’s performance in generating music that matches the emotions of a story is accessed with a user study. 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.<sup>3</sup> The user study 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 an illustrated description of the valence-arousal model of emotion and listened to 4 examples of pieces from the VGMIDI dataset labeled with the valence-arousal model. Each piece had a different emotion:

<sup>3</sup>Generated pieces can be downloaded from the following link: <https://github.com/lucasnfe/bardo-composer>

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 1-to-5 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 has been used to compare these two methods, where Group *A* of 58 participants evaluated the 5 pieces generated by Composer and another 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 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 user study. 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 referred 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.

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).

Composer outperformed the Baseline in e1-p2, e2-p1, and e5-p1. Baseline outperformed Composer e3-p1, e3-p2 and e4-p2. In the other four parts, one method performed better for valence whereas the other method performance better for arousal. 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.

## **6.4 Conclusions**

This chapter presented 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. This emotion is given as input to a Stochastic Bi-Objective Beam Search algorithm that tries to generate a piece that matches the emotion. Composer was evaluated with a user study and results 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.



## Chapter 7

# Controlling Emotions in Symbolic Music

## Generation with MCTS

### 7.1 Introduction

With the advent of reasonably large labeled datasets such as the VGMIDI dataset [40], it became feasible to investigate AAC [137] as a problem of steering the probability distribution of neural LMs towards a target emotion. Given a neural LM  $L$ , a prior sequence of music tokens  $x = [x_1, \dots, x_{t-2}, x_{t-1}]$ , a target emotion  $e$ , and a music emotion classifier  $E$ , the problem consists of decoding the  $L$ 's output layer to predict the next tokens  $x' = [x_t, x_{t+1}, \cdot, x_n]$  with the guidance of the classifier  $E$  in order to maximize the likelihood of  $x'$  having the target emotion  $e$ . The previous chapter presented SBBS, a decoding strategy that approaches this problem based on the beam search algorithm. This chapter presents a new decoding strategy to approach this problem based on Monte Carlo Tree Search (MCTS). Multiple search iterations are executed

at decoding time to build a distribution of node visits that steers the LM distribution towards a target emotion. During the search, the value of each node is computed using a music emotion classifier, and the Predictor Upper Confidence for Trees (PUCT) is used to decide the order of node expansions. The next token in the sequence is decided by sampling from this new distribution.

The search is performed over the space learned by a Music Transformer LM [60] 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 works presented in Chapter 5 [40] and Chapter 6 [39], 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. Different then these two previous works, 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 are mapped to a label.

MCTS is evaluated with two listening tests (human evaluations), 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, listeners are asked to prefer among MCTS, the validation data (human compositions), TopK sampling and SBBS [39]. In the second study, listeners are asked to annotate the emotion of pieces generated by MCTS, TopK sampling, and SBBS. Results showed that MCTS outperforms SBBS in terms of music quality while keeping the same accuracy when conveying target emotions. An expressivity analysis [123] has been performed to better understand how MCTS is conveying 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 (MT) [60] is currently one of the state-of-the-art methods for symbolic music generation and hence it is used as a base LM for MCTS. This MT 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<sup>1</sup>. The VGMIDI dataset has been used, as opposed to MAESTRO [50] or other large dataset of symbolic music, to be able to train both the language and emotion models on similar datasets. The VGMIDI pieces are encoded using a different vocabulary than the original one proposed with MT [60]. 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  $v \in [32, 36, 40, \dots, 128]$ . 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

---

<sup>1</sup><https://www.ninsheetmusic.org/>

end of all time-steps to represent the end of the piece.

This encoding scheme yields a vocabulary with 44,346 tokens. This is orders of magnitude larger than most of the other vocabularies in the literature [15]. By having more tokens, MCTS has more options to choose from at any given point of the generative process, which increases the search complexity. On the other hand, it considerably reduce the length of the encoded pieces, which reduces 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. Moreover, according to Holtzman et al. [56], LMs tend to generate less repetitive sequences with larger vocabularies.

### 7.3 Emotion Classification Model

In the previous works presented in this dissertation [40, 39], 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. A similar approach is used in this work, by fine-tuning a MT instead of an LSTM [40] or a GPT-2 Transformer [39]. MT is fine-tuned with the labeled pieces of the VGMIDI dataset. In [40, 39], 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). Each labeled piece in the VGMIDI dataset has a valence label  $v \in [-1, 1]$  and a arousal label  $a \in [-1, 1]$ . A pair of values are mapped to a categorical label (E0, E1, E2, or E3)

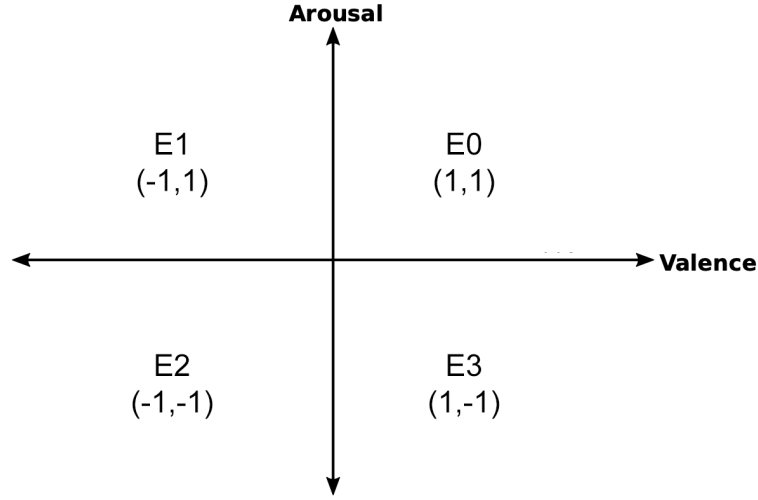


Figure 7.1: Mapping the circumplex model to a categorical model of emotion with four classes: E1, E2, E3, and E4.

by getting the quadrant in which  $(v, a)$  lie in. As shown in Figure 7.1, a piece with values  $(1, 1)$  is mapped to E0,  $(-1, 1)$  is mapped to E1,  $(-1, -1)$  is mapped to E2 and  $(1, -1)$  is mapped 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 as an emotion score, instead of a combination of two. Since MCTS evaluates the emotion of sequences of different lengths (see the detail in Section 7.4), the emotion model 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 positions of the game. MCTS variations use a variety of algorithms for deciding which branches of the tree to explore next. For example, UCT [76] uses the UCB1 formula for deciding which branches to expand in each iteration of the algorithm, while AlphaZero uses the PUCT formula [117]. This section describes how this work employs PUCT to generate music with a specific target emotion.

PUCT receives a prefix  $x = [x_0, x_1, \dots, x_l]$  of an existing musical piece as input, which will bias the process; an emotion model  $E$ ; a language model  $L$ ; a set of  $k$  symbols that is used to focus the search on the symbols with higher probability according to  $L$ ; a search budget  $d$  that defines the amount of search PUCT can perform before adding the next symbol to the musical sequence; and a target perceived 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  $n$  in the tree represents a sequence of symbols from the vocabulary. The root of the tree represents the prefix  $x$  provided as input. Each edge  $(n, n')$  from node  $n$  to node  $n'$  represents the addition of a symbol to the sequence  $n$  ( $n'$  is one symbol larger than  $n$ ). In this formulation,  $n'$  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.

**Selection:** For each node  $n$  in the PUCT, the symbol  $l$  is selected to maximize Equation 7.1, where  $Q(n, l)$  is a value computed based on the emotion model (explained in the simulation and backpropagation steps),  $c$  is an exploration constant,  $L(n, l)$  is the language model's probability of adding symbol  $l$  to the sequence  $n$ ,  $N(n)$  is the number of times node  $n$  was visited in the selection step, and  $N(n, l)$  is the number of times symbol  $l$  was chosen at node  $n$  in a selection step.

$$\arg \max_l Q(n, l) + c \times L(n, l) \times \frac{\sqrt{N(n)}}{1 + N(n, l)}, \quad (7.1)$$

In practice, only the  $k$  symbols with largest probable value according to the LM are considered in the  $\arg \max$  operator shown above. This allows the search to focus on sequences that are more promising according to the language model. PUCT recursively selects nodes in the tree according to Equation 7.1 until the symbol  $l$  at a node  $n$  leads to a sequence whose node  $n'$  is not in the PUCT tree.

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

**Simulation:** In this step, the sequence represented in the recently added node  $n'$  is evaluated according to the target emotion. The  $Q(n, l)$ -value (recall that adding  $l$  to  $n$  generated the node  $n'$ ) is given by  $E(n', e)$ . The value of  $N(n, l) = 1$  as this is the first time  $l$  is selected in node  $n$ .

**Backpropagation:** The value of  $Q(n, l)$  is used to update the  $Q$ -values of all the other node-

symbol pairs selected in the first step of the algorithm. This is achieved by following the path in the tree from the selection step in reverse order and updating the statistics of each node  $n$  and node-symbol pairs  $(n, l)$  as follows:

$$Q(n, l) = \frac{Q(n, l) \times N(n, l) + E(n', e)}{N(n, l) + 1}$$

$$N(n, l) = N(n, l) + 1.$$

The  $Q(n, l)$ -values are the average  $E$ -values of sequences with prefix given by  $n$ . The value of  $Q(n, l)$  estimates how continuations of the sequence  $n$  added to the symbol  $l$  is with respect to the target emotion. 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-symbol 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 symbol 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 symbol of the sequence. Let  $n$  be the root of the tree. The symbol  $l$  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  $n'$  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 symbol to be added to  $n'$ . This process is repeated



until the next symbol added is the end-of-music symbol. The PUCT search can be seen as an operator that changes the probability distribution over symbols given by the language model such that it accounts for the target emotion model. This is because the distribution given by  $\frac{N(n)}{\sum_l N(n,l)}$  will favor symbols that lead to pieces matching the target emotion as nodes representing such pieces are visited more often during search.

## 7.5 Experiments

MCTS is evaluated with two listening tests (user studies), one for measuring the quality of the generated pieces and one for measuring the accuracy in conveying a target emotion. All experiments were performed via Amazon Mechanical Turk (MTurk). For both experiments, MCTS is compared against SBBS, TopK sampling, and human composed pieces (from the validation data used with the emotion classifier). MCTS is not compared against Ferreira and Whitehead [40] because their approach is limited to sentiment. For each model, 10 pieces are generated with 512 tokens for each target emotion  $e \in [0, 1, 2, 3]$  (total of 160 pieces). Each model generated 40 pieces with the same set of 40 prime sequences (one prime per piece). Each prime sequence is 32 tokens long and is selected at random from the VGMIDI validation pieces with the target emotion. For the “human” model, pieces are “generated” by simply extracting the first 512 tokens of the piece with the given prime.

To generate the pieces, a MT LM is first trained with 4 layers (transformer blocks), a maximum sequence length of 2048 tokens, 8 attention heads and an embedding layer with size 384. The size of the Feed-Forward layers in the each transformer block was set to 1024. This

LM was trained with the 3640 unlabelled pieces of the extended VGMIDI dataset, where 3094 (85%) of the pieces were used for training and 546 (15%) for testing. All unlabelled pieces were 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 in by 10% [100]. The emotion classifier is then trained by fine-tuning the MT LM with an extra classification head on top. The emotion classifier was trained with the 200 labelled pieces of the VGMIDI dataset, where 140 (70%) pieces were used for training and 30 (30%) for testing. After training, the losses of the MT LM are 0.54 (training set) and 0.73 (validation set). The validation accuracy of the emotion classifier is 61%. During generation time, the LM distribution is filtered with  $k = 128$  in MCTS, SBBS and TopK. The number of beams for SBBS was set to  $b = 4$ . For MCTS, the number of simulation steps set was set to 30 and the exploration weight to 16.

### 7.5.1 Quality

In the quality listening test, a pairwise comparison is performed similar to the human evaluation in [60]. 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 human subjects 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 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 that there is a statistically significant difference between the models with  $p = 1.5e-4 < 0.01$ .

### 7.5.2 Emotion

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

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

the VGMIDI dataset [40]. 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 emotion classification model 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 very similar

to SBBS, with slightly better average accuracy. However, 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 an important results since labelling symbolic music according to emotion is a very expensive task.

<b>Model</b>	<b>E0</b>	<b>E1</b>	<b>E2</b>	<b>E3</b>	<b>Avg.</b>
<b>MCTS</b>	<b>72</b>	<b>52</b>	<b>37</b>	57	<b>54</b>
<b>SBBS</b>	67	41	30	<b>70</b>	52
<b>TopK</b>	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%), however, this was primarily due to its ability to convey the two most represented emotions in the training data (E0 and E3). These two emotions very likely are more represented in the unlabelled data as well, which was used to train the LM. Moreover, even though TopK sampling does not control emotion, the prime sequences they used to generate the pieces had the target emotions. Thus, 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.

Although MCTS performed similarly to SBBS in terms of emotion, MCTS outperformed SBBS considerably with respect to music quality. One reason for SBBS being as good as MCTS at conveying emotions is that SBBS tends to generate repetitive pieces once that maximizes both the probabilities of the language model and the emotion model. Since SBBS doesn't do backtracking, when it generates a good pattern that is likely 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 with the target emotion E2 (low valence and arousal). The MCTS piece develops the prime sequence exploring mainly the final part of the prime sequence. The generated continuation is formed by two different sections and each section is repeated once. The SBBS piece, on the other hand, simply repeats the final part of the prime sequence until the end of the piece. This is a case where SBBS repeated a given pattern that maximized the probability of the language model and the emotion model. This example also shows how backtracking allows MCTS to look for different patterns in the search space.

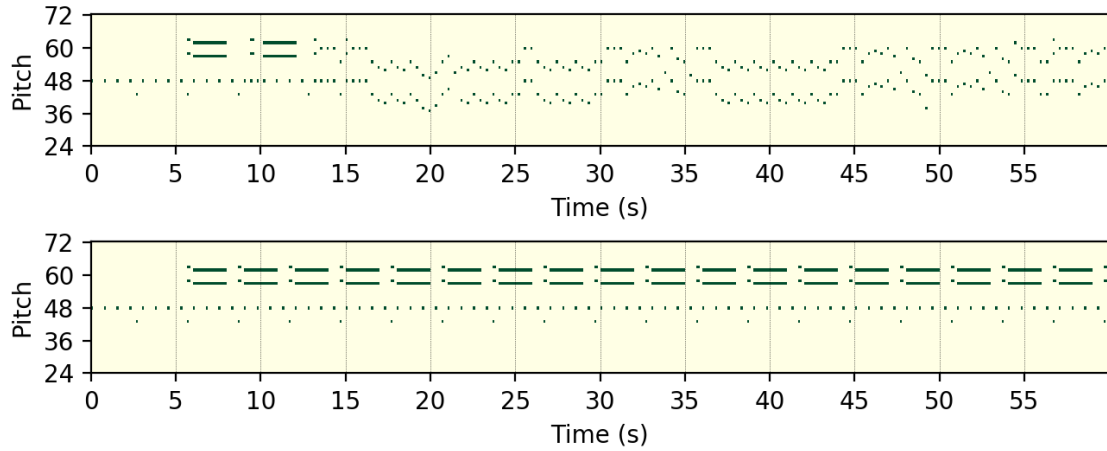


Figure 7.2: Examples of MCTS (top) and SBBS (bottom) pieces controlled to have emotion E2.

## 7.6 Expressivity Range

An expressivity analysis is performed to better understand how MCTS is conveying the different target emotions. Expressivity analysis is an evaluation method commonly used in the AI and Games research community to compare different level generators [123]. 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 and duration as the metrics to measure bias. Figure 7.3 illustrates the distribution of pitch classes and duration types 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 in label E0, however it used considerably more 8th and 32nd notes. 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 notes and 8th notes. Quarter notes are present but in a considerably lower frequency and 32nd notes are rarely used. MCTS also used a combination of 16th and 8th

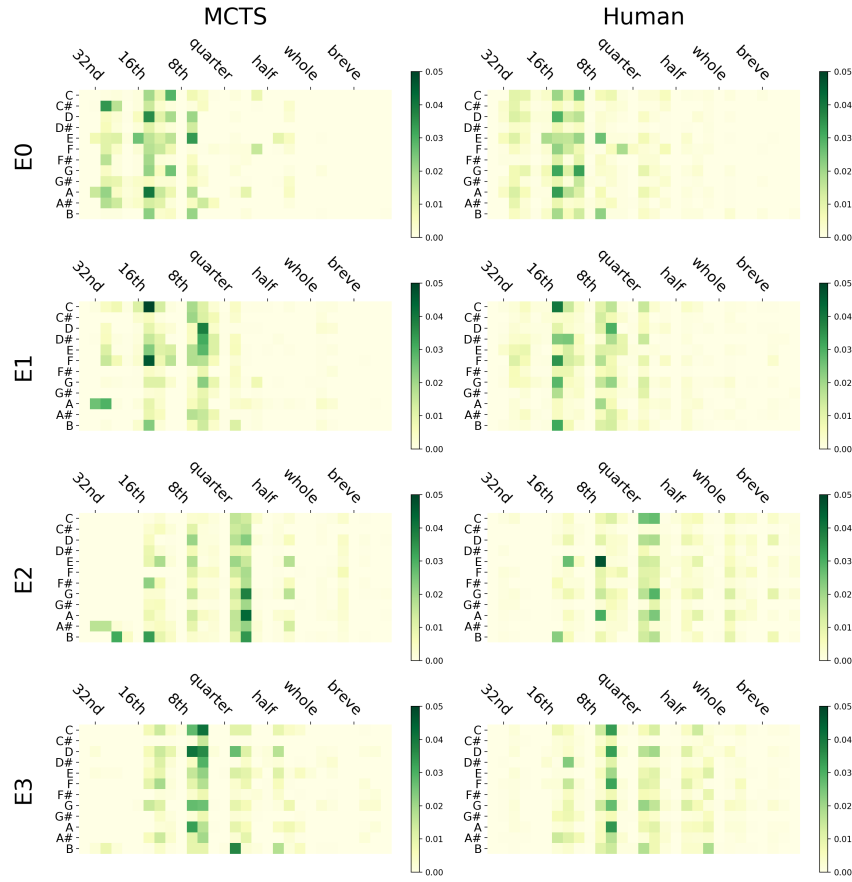


Figure 7.3: MCTS expressivity range. The x axis represents note duration in seconds and the y axis represents pitch classes.

notes, but less quarter notes and a little more 32nd notes. Label E1 represents emotions such as Tense, Angry and Frustrated. Combining these results with the expressivity range of label E0, one concludes that, VGMIDI and MCTS music 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. Pieces with this label also have a few 8th and 16th notes. Different than E0 and E1, E2 has a few longer notes such as half, whole and breve notes. When generating pieces with emotion



E2, MCTS also focused on quarter notes. However, compared with human pieces, more shorter notes were used and less longer notes were used. Label E2 encapsulates emotions such as Sad, Depressed and Tired. According to these results, human composed music as well as pieces generated by MCTS on this space tend to have rhythm patterns with longer notes.

For label E3 (high valence and low arousal), human pieces as well as generated pieces have mainly 8th notes with a few 16th, quarter and whole notes. These pieces represent emotions such as Calm, Relaxed, and Content. With these results of expressivity, one concludes that pieces with low arousal have rhythm patterns with longer notes.

## **7.7 Conclusions**

This paper applied 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 music LM, and their scores are given by an emotion classifier. A categorical model of emotion was used with four labels: high valence and arousal, low valence and high arousal, low valence and arousal, and high valence and low arousal. MCTS was evaluated with two listening tests, one to measure the quality of the generated pieces and one to measure its accuracy in conveying target emotions. A pairwise comparison is performed in the first experiment 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 generated pieces according to the proposed model of emotion to evaluate if human subjects can perceive the emotions intended by MCTS. Results showed that MCTS outperforms

SBBS in terms of quality and has slightly better accuracy when controlling emotions. With an expressivity analysis, it has been shown that MCTS generates pieces with music features similar to human compositions.

To the best of the author’s knowledge, this is the first work to apply MCTS as an algorithm to decode neural LMs. Moreover, this is the first time MCTS is being used to control emotions in generated symbolic music. Given that MCTS is agnostic to the classifier used to steer the distribution of the LM, it can be used to control different features of music that a classifier can discriminate. Since MCTS used a large vocabulary with approximately 45K tokens to generate music, its generative task is similar to other NLP generative tasks. Therefore, MCTS can also be generalized to decode and control LMs in text generation domain.

## **Chapter 8**

### **Future Work**

As discussed throughout this dissertation, the problem of composing music with a target perceived 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 [88, 129, 148, 128], 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 labelled examples with which to train the machine [81]. However, the available datasets for AAC, including the VGMIDI dataset, have only hundreds of labelled examples. Therefore,

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 labelled examples. Labelling symbolic music according to perceived 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 makes it expensive to annotate symbolic music pieces according to perceived emotion because 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 [79] where users would have external incentive (other than money) to collaborate in the task of labelling music with a target emotion.

The VGMIDI dataset and most of the other AAC datasets are limited to piano music. Thus, a great new contribution would be to create datasets with polyphonic music with multiple instruments. This would allow one to use NNs to model the relationship between different timbres and perceived emotion. One of the major challenges in creating large datasets with multiple instruments is to collect a large collection of pieces that use exactly the same set of instruments. Typically, the symbolic music datasets are created by gathering MIDI files publicly available from the Internet. Since the MIDI files come from different sources in the Internet, it is hard to guarantee consistency among the files. Once again, this could be addressed by making 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 encountered with the VGMIDI dataset is that negative video game piano pieces are considered ambiguous by human subjects. Thus, another future work would be to create a dataset of pieces that are less ambiguous. One way to address this would be

to use movie soundtracks and ask annotators to label the pieces together with their respective scenes, for example. The problem with building soundtrack datasets is that most of this music is not publicly available in the symbolic format. One way to solve this problem would be to ask professional composers to create scores for a set of video clips that represent different emotions.

## 8.2 Music Representation

Most datasets of symbolic music are organized as MIDI files. To learn models from these files one has to create a vocabulary of music tokens extracted from the MIDI events or from the piano roll representation of that MIDI. This dissertation explored vocabularies with different sizes, however, it is still unclear how different vocabularies affect the quality and controllability of the generated music. This analysis should be performed with different decoding strategies and different data regimes (low data vs. big data). For example, one can investigate the impact of vocabulary size in the level of repetition in generated music. Vocabulary size can also impact the quality of representations learned by VAEs. Another possible future direction related to music representation is to learn representations from MIDI files. For example, one could use a *vector quantised variational auto encoder* (VQ-VAE) to learn a vocabulary from MIDI files. Having better music representation has the potential to increase the overall quality of generated music and support better model controllability, which are key components of neural AAC.

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 of 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. This problem has been actively explored by the deep learning community [99, 90, 142], but the state-of-the-art methods still generate music that significantly far from human-composed music [26]. Designing neural networks 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 consists of fine-tuning a LM with a genetic algorithm and the other two are decoding strategies that steer the LM’s distribution in 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 could investigate how to control GANs and VAEs to compose music with controllable emotion, or design completely new architectures for that purpose. GANs have shown great potential in music generation [95] and controllability through conditioning []. Thus, one could 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 to use GANs for AAC is that training them with too little data typically leads to discriminator overfitting, causing training to diverge [68]. Regarding VAEs, a future work can be to extend Music FaderNets [128] to learn a representation between low-level music structure and different emotions.

Another interesting direction of future work consists of fine-tuning generative models (e.g. transformer) with reinforcement learning. This could 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 training procedure 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 great potential as decoding strategies to control emotion in music generated by LMs. As a future work, one could 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 could investigate how to improve the quality of the music generated by both these methods. For example, exploring how to reduce repetition in the music generated by SBBS. This could be done by training a repetition discriminator that models a common level of repetition in human-composed music [56]. This discriminator should be combined with the emotion discriminators in decoding time. One could also apply trainable weights to each of these discriminators in order linearly adjust their contributions to the final generated music.

Section 4.2 showed different strategies to decode neural LMs in the natural language domain. Exploring these approaches to control emotion in generated music are another important direction of future works. For example, one could explore how to use the Plug and Play

LM [22] to compose music with controllable emotion.

## 8.5 Applications

This dissertation showed some progress with Bardo Composer towards generating affective music in real time for tabletop roleplaying games. Bardo Composer was evaluated on the task of scoring videos of a D&D campaign. Thus, a future work related to Bardo Composer consists of building a prototype to score 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 [101]. Mapping the 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 could also build upon Bardo Composer to generate soundtracks for other media. For example, Bardo Composer could be applied to score audio books with the same approach used to score tabletop roleplaying games. One can extract the sentiment or emotion from video games scenes based on game events (e.g. powerup pickup) and use 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 soundtracks for movies. Another interesting 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 users conversations.

## Chapter 9

### Conclusion

This dissertation has presented three different search-based approaches to control music language models to generate symbolic music with a target perceived emotion. The first approach, described in Chapter 5, consists of a genetic algorithm to fine-tune a LSTM language model towards generating negative and positive pieces [40]. This approach showed to be successful in generating positive pieces, but negative pieces were considered slightly ambiguous, according to human evaluators. 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 [39]. 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 monte carlo tree search 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 [118] and 3,640 unlabelled ones, where 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 roleplaying games. Bardo Composer uses a speech recognition system to translate player speech into text, which is classified according to a model of emotion. Bardo Composer then uses SBBS with a neural model to generate musical pieces conveying the desired emotion.

The contributions of this dissertation showed that combining neural LMs with search methods has strong potential in generating music with controllable emotion. As discussed in Chapter 8, there are several possibilities of future work, from increasing the VGMIDI dataset to developing new neural models and applying Bardo Composer to other media (e.g. video games and films). Hopefully, the ideas discussed in this dissertation will inspire others to explore some of this future work and other ideas that will allow neural models to generate music with human level quality.

## 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] Philip Ball. Making music by numbers online, 2005.
- [7] 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.
- [8] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The journal of machine learning research*, 3:1137–1155, 2003.
- [9] 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.
- [10] Karl Bergström and Staffan Björk. The case for computer-augmented games. *Transactions of the Digital Games Research Association*, 1(3), 2014.
- [11] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. The million song dataset. 2011.
- [12] John Biles et al. Genjam: A genetic algorithm for generating jazz solos. In *ICMC*, volume 94, pages 131–137. Ann Arbor, MI, 1994.
- [13] 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.

- [14] 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.
- [15] Jean-Pierre Briot, Gaëtan Hadjeres, and François Pachet. Deep learning techniques for music generation-a survey. *arXiv preprint arXiv:1709.01620*, 2017.
- [16] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] 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>.
- [21] David Cope. Experiments in musical intelligence (emi): Non-linear linguistic-based composition. *Journal of New Music Research*, 18(1-2):117–139, 1989.
- [22] 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.
- [23] 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.
- [24] 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.
- [25] A. K. Dewdney. Computer recreations. *Scientific American*, 261(2):102–106, 1989. ISSN 00368733, 19467087. URL <http://www.jstor.org/stable/24987368>.
- [26] 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.

- [27] Chris Donahue, Huanru Henry Mao, and Julian McAuley. The nes music database: A multi-instrumental dataset with expressive performance attributes. In *ISMIR*, 2018.
- [28] 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.
- [29] 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.
- [30] MO Duff. Backpropagation and bach’s 5th cello suite (sarabande). In *International 1989 Joint Conference on Neural Networks*, pages 575–vol. IEEE, 1989.
- [31] Kemal Ebcioglu. An expert system for harmonizing four-part chorales. *Computer Music Journal*, 12(3):43–51, 1988.
- [32] 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.
- [33] Panteleimon Ekkekakis. *The measurement of affect, mood, and emotion: A guide for health-behavioral research*. Cambridge University Press, 2013.
- [34] Paul Ekman. An argument for basic emotions. *Cognition & emotion*, 6(3-4):169–200, 1992.
- [35] 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.
- [36] B Eno. Generative music, speech at the imagination conference, 1996.
- [37] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*, 2018.
- [38] Andy Farnell. An introduction to procedural audio and its application in computer games. In *Audio mostly conference*, volume 23. Citeseer, 2007.
- [39] 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.
- [40] 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>.

- [41] Anders Friberg, Roberto Bresin, and Johan Sundberg. Overview of the kth rule system for musical performance. *Advances in cognitive psychology*, 2(2):145, 2006.
- [42] Martin Gardner. Mathematical games. *Scientific American*, 223(4):120–123, 1970. ISSN 00368733, 19467087. URL <http://www.jstor.org/stable/24927642>.
- [43] Stanley Gill. A technique for the composition of music in a computer. *The Computer Journal*, 6(2):129–133, 1963.
- [44] Jon Gillick, Kevin Tang, and Robert M Keller. Machine learning of jazz grammars. *Computer Music Journal*, 34(3):56–66, 2010.
- [45] Michael Good. Musicxml for notation and analysis. *The virtual score: representation, retrieval, restoration*, 12(113-124):160, 2001.
- [46] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [47] 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.
- [48] 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.
- [49] 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.
- [50] 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=r11YRjC9F7>.
- [51] 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.
- [52] 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>.

- [53] 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.
- [54] 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.
- [55] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [56] 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.
- [57] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.
- [58] SR Holtzman. Using generative grammars for music composition. *Computer Music Journal*, 5(1):51–64, 1981.
- [59] 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.
- [60] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinulescu, and Douglas Eck. Music transformer. *arXiv preprint arXiv:1809.04281*, 2018.
- [61] Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Douglas Eck. Counterpoint by convolution. *arXiv preprint arXiv:1903.07227*, 2019.
- [62] Yu-Siang Huang and Yi-Hsuan Yang. Pop music transformer: Generating music with rhythm and harmony. *arXiv preprint arXiv:2002.00212*, 2020.
- [63] Bruce Jacob. Composing with genetic algorithms. 1995.
- [64] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.
- [65] 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.
- [66] 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.



- [67] Kevin Jones. Compositional applications of stochastic processes. *Computer Music Journal*, 5(2):45–61, 1981.
- [68] 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.
- [69] 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.
- [70] 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.
- [71] Sunjung Kim and Elisabeth André. Composing affective music with a generate and sense approach. In *FLAIRS Conference*, pages 38–43, 2004.
- [72] 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.
- [73] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.
- [74] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [75] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [76] 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.
- [77] Ben Krause, Iain Murray, Steve Renals, and Liang Lu. Multiplicative LSTM for sequence modelling. *ICLR Workshop track*, 2017.
- [78] Peter Langston. Six techniques for algorithmic music composition. In *Proceedings of the International Computer Music Conference*, volume 60. Citeseer, 1989.
- [79] Edith Law, L. V. Ahn, R. Dannenberg, and Mike Crawford. Tagatune: A game for music and sound annotation. In *ISMIR*, 2007.
- [80] 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.

- [81] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.
- [82] Fred Lerdahl and Ray S Jackendoff. *A Generative Theory of Tonal Music, reissue, with a new preface*. MIT press, 1996.
- [83] 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.
- [84] 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.
- [85] 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.
- [86] Sergio Luque. The stochastic synthesis of iannis xenakis. *Leonardo Music Journal*, 19: 77–84, 2009.
- [87] Qi Lyu, Zhiyong Wu, Jun Zhu, and Helen Meng. Modelling high-dimensional sequences with lstm-rtrbm: Application to polyphonic music generation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
- [88] Rishi Madhok, Shivali Goel, and Shweta Garg. Sentimozart: Music generation based on emotions. In *ICAART (2)*, pages 501–506, 2018.
- [89] 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.
- [90] 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.
- [91] 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.
- [92] Eduardo Reck Miranda. Cellular automata music: An interdisciplinary project. *Journal of New Music Research*, 22(1):3–21, 1993.
- [93] 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.
- [94] James Anderson Moorer. Music and computer composition. *Communications of the ACM*, 15(2):104–113, 1972.

- [95] 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.
- [96] Gerhard Nierhaus. *Algorithmic composition: paradigms of automated music generation*. Springer Science & Business Media, 2009.
- [97] 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.
- [98] Christopher Olah. Understanding lstm networks. 2015.
- [99] 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.
- [100] 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.
- [101] 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.
- [102] Rafael R Padovani, Lucas N Ferreira, and Levi HS Lelis. Bardo: Emotion-based music recommendation for tabletop role-playing games. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.
- [103] 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.
- [104] George Papadopoulos, Geraint Wiggins, et al. A genetic algorithm for the generation of jazz melodies. *Proceedings of STEP*, 98, 1998.
- [105] W Gerrod Parrott. *Emotions in social psychology: Essential readings*. psychology press, 2001.
- [106] 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.
- [107] 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.

- [108] David L Poole and Alan K Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.
- [109] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.
- [110] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. In *Arxiv*, 2018.
- [111] 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.
- [112] Colin Raffel. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. PhD thesis, Columbia University, 2016.
- [113] 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.
- [114] 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](https://nips2017creativity.github.io/doc/Hierarchical_Variational_Autoencoders_for_Music.pdf).
- [115] 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.
- [116] Joseph Rocca. Understanding variational autoencoders (vae). *Data Science*, 2019.
- [117] Christopher D. Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, March 2011.
- [118] James A Russell. A circumplex model of affect. *Journal of personality and social psychology*, 39(6):1161, 1980.
- [119] Harald Schmidl. Pseudo-genetic algorithmic composition. In *GEM*, pages 24–29. Cite-seer, 2008.
- [120] 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.
- [121] Thalles Silva. An intuitive introduction to generative adversarial networks (gans). *freeCodeCamp. org*, 2018.
- [122] Mary Simoni. *Algorithmic composition: a gentle introduction to music composition using common LISP and common music*. Michigan Publishing, University of Michigan Library, 2003.

- [123] 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.
- [124] 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.
- [125] 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>.
- [126] Makis Solomos. Cellular automata in xenakis’s music. theory and practice. In *International Symposium Iannis Xenakis (Athens, May 2005)*, pages 11–p, 2005.
- [127] Simeng Sun and Mohit Iyyer. Revisiting simple neural probabilistic language models. *arXiv preprint arXiv:2104.03474*, 2021.
- [128] 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.
- [129] Xiaodong Tan and Mathis Antony. Automated music generation for visual art through emotion. In *ICCC*, pages 247–250, 2020.
- [130] Sever Tipei. *MPI: a computer program for music composition*. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 1975.
- [131] Peter M Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43, 1989.
- [132] 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.
- [133] 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.
- [134] 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].
- [135] Chris Walshaw. *Abc2mtex: An easy way of transcribing folk and traditional music*, version 1.0. *University of Greenwich, London*, 1993.

- [136] Duncan Williams, Alexis Kirke, Joel Eaton, Eduardo Miranda, Ian Daly, James Hollowell, 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.
- [137] 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.
- [138] 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.
- [139] Stephen Wolfram. *A new kind of science*, volume 5. Wolfram media Champaign, IL, 2002.
- [140] 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.
- [141] Iannis Xenakis. *Formalized music: thought and mathematics in composition*. Number 6. Pendragon Press, 1992.
- [142] 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.
- [143] 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.
- [144] 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.
- [145] 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.
- [146] 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.
- [147] Marcel Zentner, Didier Grandjean, and Klaus R Scherer. Emotions evoked by the sound of music: characterization, classification, and measurement. *Emotion*, 8(4):494, 2008.

- [148] Kun Zhao, Siqu 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.
- [149] 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.
- [150] 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.