UNIVERSITY OF CALIFORNIA SAN DIEGO

From Notes to Musical Form: A Machine Learning Approach

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

in

Music

by

Lilac Atassi

Committee in charge:

Professor Shahrokh Yadegari, Chair
Professor Garrison W. Cottrell
Professor Thomas Erbe
Professor Miller S. Puckette
Professor Robert Wannamaker

2024

The Dissertation of Lilac Atassi is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2024

TABLE OF CONTENTS

Dissertation Approval Page iii

Table of Contents iv

List of Figures vii

Acknowledgements xii

Vita xiii

Abstract of the Dissertation xiv

Chapter 1 Introduction 1

Chapter 2 ML Music Generation 4
2.1 Abstract 4
2.2 Introduction 4
2.3 Data 9
2.3.1 Data representation 9
2.3.2 Publicly available data 11
2.4 Applications 12
2.4.1 Music transcription 13
2.4.2 Co-composition 13
2.4.3 Music production 15
2.4.4 Synthetic Instrument 16
2.5 Models and network architectures 16
2.5.1 Feedforward networks 16
2.5.2 Recurrent networks 20
2.5.3 Generative adversarial networks 21
2.5.4 Autoencoders 22
2.5.5 Transformers 23
2.5.6 Diffusion 25
2.6 Training methods 26
2.6.1 Optimization 27
2.6.2 Autoregressive order 30
2.7 Evaluation approaches 30
2.8 Beyond statistical models 32
2.9 Experiments with diffusion 35
2.10 Conclusions 38

Chapter 3 Diffusion Models 41
3.1 Abstract 41
3.2 Introduction 41

3.3 Deep Unsupervised Learning using Nonequilibrium Thermodynamics 43
3.4 Modern Diffusion Models 47
3.5 Conditional sampling and latent space 51
3.6 Neural network architecture 52
3.7 What the network perceives 57
3.8 Experiments 59
3.9 Research direction 63
3.10 Conclusions 65

Chapter 4 Large Language Models:
From Notes to Musical Form 66
4.1 Abstract 66
4.2 Introduction 66
4.3 Unlearnable Musical Form 69
4.4 EnCodec and MusicGen 71
4.5 Large Language Models 74
4.6 Controlling MusicGen by a Language Model 76
4.7 Evaluation 81
4.7.1 Objective Evaluation 82
4.7.2 Subjective Evaluation 86
4.8 Meta Optimization by LLM 87
4.9 Conclusions 91

Chapter 5 Dimensionality Reduction 93
5.1 Abstract 93
5.2 Introduction 93
5.3 Dimensionality reduction algorithms 96
5.3.1 Linear methods 96
5.3.2 Non-linear methods 98
5.4 Dimensionality reduction algorithms in musical interfaces 112
5.4.1 MusicMappr musical interface 113
5.4.2 the Infinite drum machine musical interface 114
5.4.3 Text-to-music interface 117
5.5 Conclusions 118

Chapter 6 Allocentric and Egocentric Controllers: Similarities and Differences 119
6.1 Introduction 119
6.1.1 Gestural Controller 121
6.1.2 Allocentric coordinates 121
6.1.3 Egocentric coordinates 123
6.1.4 Effort-based mapping 125
6.1.5 Affordances and Constraints 125
6.1.6 Discussion 127

Chapter 7 Human Voice Detection For Search and Rescue Operations 129
7.1 Introduction 129
7.2 Data 130
7.3 Performance metric 131
7.4 Experiment one: convolutional neural network 131
7.5 Experiment two: ResNet model 133
7.6 Experiment three: ResNet model with OpenL3 transfer learning 136
7.7 Discussion 138

Chapter 8 Conclusion 140

Bibliography 141

LIST OF FIGURES

Figure 2.1. The steps of Gibbs sampling done by Coconet. 19

Figure 2.2. Hand images generated using Stable Diffusion. 35

Figure 2.3. Two 16-beat long pieces generated by the model. Each piano-roll is linked to a synthesized performance by GarageBand on YouTube, you can listen by clicking on the images. 37

Figure 2.4. Two 32-beat long pieces generated by the model. Each piano-roll is linked to a synthesized performance by GarageBand on YouTube, you can listen by clicking on the images. 37

Figure 2.5. A 160-beat long piece generated by the model. The piano-roll is linked to a synthesized performance of about a minute and a half by GarageBand on YouTube, you can listen by clicking on the images. 38

Figure 3.1. The approach used by diffusion models to learn the distribution of the data. 42

Figure 3.2. Demonstrating the forward and backward diffusion processes. 46

Figure 3.3. Demonstrating the conversion of a distribution to another analytical distribution. 47

Figure 3.4. The difference between linear and cosine forward diffusion weighting. Source: Reproduced from [1], licensed under CC BY 4.0. 49

Figure 3.5. Left, the original sampling process denoises a sample drawn from the prior distribution in each step. Right, the sampling process that alternates between estimating the noiseless point from the data distribution, and adding some noise less than the previous step. 50

Figure 3.6. Demonstrating image inpainting using a diffusion model. 51

Figure 3.7. The UNet architecture used in my experiments. 53

Figure 3.8. The top figure shows a tensor in blue and a convolution filter in orange, applying the conv filter, that is overlapping the filter tensor on the input tensor and multiplying them element-wise, and then shifting the filter and repeating, results in a single matrix or a 2d tensor shown in green. 54

Figure 3.9. The figure depicts down-sampling a feature-map by using conv filter with stride of 2. 55

Figure 3.10. The upsampling operation used in UNet is based on simple nearest neighbor interpolation. This is simply duplicating each element in the input feature-map 3 times to double the number of rows and columns of the output feature-map. 56

Figure 3.11. Using AM, the input that maximizes the activation of three convolution filters at the lowest resolution of the UNet trained on piano-rolls are visualized. 58

Figure 3.12. Top: original piano-roll segment. Middle and bottom: the diffusion model is prompted with the first half of the original piano-roll and the second half is generated by the model. 60

Figure 3.13. Top: original piano-roll segment. Bottom: the model is prompted with the high register pitch and the low register pitch is generated by the model. Each piano-roll in this figure is linked to the synthesized audio. 60

Figure 3.14. Top: original piano-roll segment with the first and last quarter used as prompts to the diffusion model. Bottom: the model fills the middle half of the piano-roll to be coherent with the first and last quarter. Each piano-roll in this figure is linked to the synthesized audio. 61

Figure 3.15. Top: original piano-roll segment. Bottom: a variation of the original piano-roll generated by the diffusion model. Each piano-roll in this figure is linked to the synthesized audio. 61

Figure 3.16. Top and middle: two original piano-roll segments that are used to generate a new piano-roll that sounds similar to both. Bottom: the diffusion model is used to interpolate a piano-roll between the two original piano-rolls. . . . 62

Figure 3.17. Reducing the size of a piano-roll is possible by reducing the number of rows, and indicating what MIDI note each element of the piano-roll play. -1 indicates the note does not play any note. 63

Figure 3.18. Depicting the two step process to generate a complete piece with coherent form. 63

Figure 4.1. Illustrating the incoherence in images generated by Dall-E 3 (https://chatgpt.com/), Midjourney (https://midjourney.com/), and Meta AI (https://meta.ai/). 70

Figure 4.2. Visualizing the fused self similarity matrices [2] of three songs generated by MusicGen, left column, and three songs generated by my method, right column. 82

Figure 4.3. Visualizing the self similarity matrices for 3 MusicGen samples, one sample from my method, and one from Pond5. 83

Figure 4.4. The mean (top row) and the variance (bottom row) of the fused self-similarity (SS) matrices are estimated with 100 samples from Pond5, generated by my method, and by MusicGen. 84

Figure 4.5. Left: The subjective comparison of the generated music and sampled from Pond5 by non-musicians is measured through the MOS, based on how engaging the music is. Right: the subjective comparison of the samples by musicians, critiquing the musical structures. 85

Figure 4.6. The PO-LLM proposes new instructions and few-shot samples for the MP-LLM. 87

Figure 4.7. The average MOS of the prompts generated in the exploration phase of the optimization method. 90

Figure 4.8. The average MOS of the top 5 prompts in each iteration of the optimization. 91

Figure 5.1. Depicting the components of a digital musical instrument. The mapping engine and sound engine together are considered the core of the instrument and define the characteristics of the instrument. Adopted from [3]. 94

Figure 5.2. Left, the 400 points mapped onto 2d using vanilla t-SNE. The colors represent the vertex of the tetrahedron each point belongs to. Right: the KL divergence cost over 5,000 optimization steps. 104

Figure 5.3. Left, the 400 points mapped onto 2d using my proposed method. The colors represent the vertex of the tetrahedron each point belongs to. The four clusters are placed at the four vertices of a square in 2d using my method. Right: the KL divergence cost over 600 optimization steps. 104

Figure 5.4. Left, the points in 2d mapped using my method at the optimization step 300. Right: the KL divergence cost function value computed for the offset positions of a control point, the orange point at the bottom left in the left subplot. 105

Figure 5.5. Left, the projected points onto 2d by vanilla t-SNE. The boundary between the cluster is observably fuzzy. Right, the cost value at step 100 is about 10% larger than the minimum value at step 5,000. 106

Figure 5.6. Left, the mapped clusters by my method. Despite the fuzzy boundaries, the clusters have the desired layout. Note the non-trivial transformation between this layout and the layout in Fig. 5.5. 106

Figure 5.7. A dataset of 28 by 28 pixel images of handwritten digits is mapped to a two dimensional space, using (a) PCA, (b) an autoencoder, (c) parametric t-SNE. 110

Figure 5.8. Inverse Autoregressive Flow changes the distribution of the points in the latent space to resemble the prior distribution more closely. Source: Reproduced from [4], licensed under CC BY 4.0. 110

Figure 5.9. The proposed approach in [5] using a VAE to map the control values from a 75-dimensional space to a 16-dimensional space. Source: Reproduced from [5], licensed under CC BY 4.0. 113

Figure 5.10. The interface of the infinite drum machine. Each point on the screen corresponds to a sound clip. Selecting four points for the four channels of the drum machine, the sequencer can also be used to modify the rhythm for each channel. 114

Figure 5.11. The fingerprints of 16 audio files from the infinite drum machine. Each fingerprint is a 32 by 32 matrix. 115

Figure 5.12. One point from each sound cluster is controlled by Hinged t-SNE to have a defined location for each cluster. A short video of interacting with the interface is available at https://youtu.be/rGw3PUAd6a8. 116

Figure 5.13. Using Hinged-TSNE to arrange the prompts on the screen, with the tempo of the pieces increasing from left to right. The user can explore the prompts to understand, for instance, what level of musical description details is understood by the text-to-music model. 117

Figure 6.1. Left: The egocentric reference frame is attached to the tracked chest point. The red line is the distance between the wrist to the origin in the 3d space. Right: the allocentric reference frames are attached to points in space, one for each arm. 122

Figure 7.1. Visualizing the relationship between frequency and mel scale bins. 132

Figure 7.2. Visualizing the spectrogram of an audio clip containing voice and background noise. 132

Figure 7.3. The convnet architecture. 134

Figure 7.4. The ResNet architecture. 135

Figure 7.5. The L3 network head architecture. 137

Figure 7.6. The ROC curves for the three experiments. Regular convolutional network performs the worst. The ResNet model performs significantly better. A ResNet on the top of the L3 embeddings has promising performance and outperforms the other two models. 138

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Shahrokh Yadegari for his support as chair of my committee. His advice and knowledge shaped the direction of my research. I would also like to thank my committee members: Garrison W. Cottrell, Thomas Erbe, Miller Puckette, and Robert Wannamaker for their significant guidance and contributions to my work and this dissertation. Special appreciation goes to my husband and daughter for their love and support throughout this journey.

Chapter 3, in part, is a reprint of the material as it appears in Sound and Music Computing Conference Proceedings 2023, Atassi, Lilac. The dissertation/thesis author was the primary investigator and author of this paper.

Chapter 5, in part, is a reprint of the material as it appears in International Conference on New Interfaces for Musical Expression Proceedings 2022, Atassi, Lilac. The dissertation/thesis author was the primary investigator and author of this paper.

Chapter 6, in part, is a reprint of the material as it appears in Leonardo Music Journal, Atassi, Lilac, MIT Press 2022. The dissertation/thesis author was the primary investigator and author of this paper.

VITA

2017 BFA in Piano Performance, California Institute of the Arts

2019 MFA in Composition, California Institute of the Arts

2024 PhD in Music, University of California San Diego

PUBLICATIONS

Atassi, L. 2023. "Generating Symbolic Music Using Diffusion Models." Sound and Music Computing Conference.

Atassi, L. 2022. "Hinged t-SNE for Musical Interfaces." The Doctoral Consortium at the International Conference on New Interfaces for Musical Expression.

Atassi, L. 2022. "Allocentric and Egocentric Controllers: Similarities and Differences." Leonardo Music Journal.

ABSTRACT OF THE DISSERTATION

From Notes to Musical Form: A Machine Learning Approach

by

Lilac Atassi

Doctor of Philosophy in Music

University of California San Diego, 2024

Professor Shahrokh Yadegari, Chair

Generative Machine Learning (ML) models can create text, images, videos, audio, and music. These models are valuable in tools for music co-composition and co-production, as well as for enabling non-musicians to engage in the music-making process. My research focuses on understanding the ML approach to music generation, centering on two key questions: What are the limitations of current generative music models? Can incremental improvements, without introducing a new class of generative models, address these limitations?

My dissertation examines various techniques, challenges, and applications of ML algorithms for music generation. The research includes projects on musical interfaces for generative models, symbolic music generation, and music generation with a focus on form. Additionally,

three chapters report on my research into dimensionality reduction, performative controllers, and human voice activity detection, offering further insight into key technological advancements relevant to these areas. Given the rapid pace of algorithmic development, many of the specific methods proposed may soon become outdated. Therefore, I believe the most significant outcome of my research is the identification of the challenges faced by autonomous ML-based music generation models and the development of general solutions to these challenges.

In particular, my dissertation presents a novel denoising diffusion probabilistic model for symbolic music generation, offering improved computational efficiency compared to other diffusion methods in the literature. While several ML-based music generation techniques, including my diffusion-based approach, have been explored, they all face limitations due to the small context window, typically capped at one minute. The primary challenge in this field is generating long-form music that rivals human-composed music in terms of form and structure. After hypothesizing why extending the context window alone cannot solve the problem of structural diversity in generated long-form music, due to combinatorial variability, I introduce my generalized approach. Subjective and objective metrics demonstrate a meaningful improvement in musical form compared to the current generation of generative music models. While this work shows promise, achieving models that match the quality of great composers remains an exciting and open challenge.

Chapter 1

Introduction

Computer technology has revolutionized music, from composition to live performances, with the advancement of both software and hardware over time. Machine learning (ML) has gained more attention in the past decade due to improved methods and more powerful computers. ML is considered an innovation with potential impacts across various fields. ML models learn to perform tasks by analyzing large datasets and identifying patterns, enabling them to operate without the need for explicit, predefined instructions. This approach, in which computers learn from examples, enables them to perform tasks too complex to be solved through direct programming.

A specific category of ML, known as generative models, is designed to learn and understand the underlying distribution of data samples. This understanding allows these models to generate new samples closely resembling the original data while preserving inherent patterns and characteristics. For instance, designing algorithms that generate music from vague descriptions provided by non-musicians is a challenging task due to the subjective and abstract nature of such descriptions. However, by training on pairs of music samples and their corresponding descriptions, an ML model can learn to interpret these descriptions and accurately convert them into music.

With the capabilities of generative machine learning, computers can now take on music-related tasks that were once deemed unachievable. One area of research that exemplifies this

is co-composition, where composers work alongside generative models to create music. This collaborative approach not only inspires composers by introducing novel musical ideas but also has the potential to significantly reduce the effort required to develop new compositions.

Highly controllable generative models that require minimal guidance can be applied across various music applications, such as co-composition, music transcription, and music production. My research focuses on developing methods to enhance control while reducing the need for extensive guidance in generative models. Autoregressive models are commonly used due to their lower computational requirements during training and inference. However, their rigid sequence generation process limits their suitability for tasks such as infilling music scores (which is discussed in Chapter 2), which demand more flexible inference methods. In contrast, diffusion models offer greater flexibility, as they do not rely on a fixed direction for inference, enabling them to effectively fill gaps in a music score. A central question in my dissertation is how to adapt a diffusion model to efficiently generate symbolic music. My research introduces a method specifically designed to efficiently infill symbolic music scores, addressing a critical challenge in the field.

The second major focus of my research on music generative models is the issue of musical form. While current models can generate music of arbitrary length, pieces that extend beyond a minute often lack coherent structure, which is essential for creating meaningful and engaging compositions. In other words, music generated by these models that exceeds two minutes tends to meander, revealing a significant lack of structural design. To address this limitation, my research explores a novel approach that integrates two models to generate music using a top-down method—first outlining the overall musical form and its constituent parts, and then filling in the music for each section.

The third key aspect of my research on generative models centers on the control and guidance of these systems. Recent advancements in generative models have introduced new interfaces that demand fresh, innovative designs for effective integration into larger systems. Specifically, I propose a dimensionality reduction method that refines and guides the genera-

tive process, enhancing the model's ability to produce structured musical compositions. This method is designed to improve the model's capacity to generate music that not only adheres to predetermined forms but also aligns with the desired aesthetic and structural qualities.

Music generative models are at an inflection point. Tasks in music composition and production that were once considered beyond the reach of computers are now yielding promising results due to advances in these models. While further progress is still required, generative models are beginning to demonstrate their potential to significantly influence the ways in which music is composed and produced. In this dissertation, I take a forward-looking approach, positioning my research at the forefront of efforts to address these emerging challenges in the field of music generative models.

Chapter 2 provides an overview of the use of ML in music generation, including its applications, methods, and datasets. Chapter 3 introduces diffusion models and presents my work on diffusion-based music generation. Chapter 4 reviews large language models (LLMs) and discusses my method for integrating LLMs with music generative models to generate music with form. My work on musical interfaces is discussed in Chapters 5 and 6. Chapter 7 presents the final report of the STTR project to which I contributed. Finally, Chapter 8 concludes the dissertation with a summary of my research.

Chapter 2

ML Music Generation

2.1 Abstract

This chapter reviews the literature on Machine Learning based methods for music generation. First, I discuss how machine learning methods learn from data, and the ways machine learning is used for music generation. Then, some applications of music generative models are reviewed. The common artificial neural network architectures, training methods and evaluation approaches are discussed later in this chapter. I present and subjectively evaluate some generated samples from my experiments with diffusion models in the section before last. The last section presents my concluding discussion and research opportunities that I find interesting.

2.2 Introduction

While there have been many new developments in Machine Learning, particularly in generative machine learning, these newer methods are still mostly based on the methods that were developed decades ago. Therefore, to understand these methods it is beneficial to briefly review some relevant concepts.

Before the term machine learning was coined by Arthur Samuel in 1959, while he was working on developing algorithms for the game of checkers [6], the broader term Artificial Intelligence (AI) was used to describe various problems researchers were tackling. AI is commonly defined as enabling a computer system to perform tasks that typically require human

intelligence, such as speech recognition. Machine Learning (ML), a branch of AI, focuses on algorithms that improve through experience and training data. One of the earliest examples of ML is Samuel's checkers program, which learned to play by competing against itself and eventually surpassed Samuel's own skill. This success demonstrated how machines could learn new tasks without explicit programming, inspiring more researchers to explore the potential of machine learning.

A machine learning algorithm is used to train a model. The ML model, then consequently, is used to do inference. Given the machine learning algorithm learns from data, the data used to train the model is one of the important ingredients for the success of the ML models. Based on how training data is prepared for a task on hand, there are three main types of Machine Learning algorithms:

1. Supervised learning: The model is given a set of input and output data pairs, and it learns by the provided examples. At the inference time, the model is given an input data point and the model predicts the output. The model is expected to generalize to unseen data.

2. Unsupervised learning: The model consists of unlabeled and unstructured input data, and it learns by identifying patterns and correlations in the input data. Most of the ML models reviewed in this paper belong to this category, since generative models in general learn to produce new data points that are similar to the given.

3. Reinforcement learning: the model learns from experience and interaction with the environment and a numerical reward provided by the environment after a sequence of actions are completed. There is no direct access to the target or key answers.

ML models have been successfully used in a wide range of applications, from playing games like chess and go, to self-driving cars and text translation. The ML models based on the application can be divided into two categories:

1. Predictive models are used to predict the output for some unseen data. Almost always the predicted value can be compared with the true value to measure the amount of error by the model. For example, a model may predict some stock share value a day ahead, and after a day

passes it is possible to measure how accurate the prediction was.

2. Generative models are trained to estimate the probability distribution that the training data is drawn from. Then the estimated probability distribution is used to generate new samples. A more formal description is presented at the end of this section. The reviewed ML models of main interest in this paper fall in this category. For instance, generative models can write a novel, generate an image or a piece of music.

Generative models for music can be designed either to generate symbolic music or audio. Symbolic music, for example could be in the form of piano rolls, midi or sequence of events. A synthesizer or an acoustic instrument is always needed to convert the symbolic music to an audible form. Audio generative models can generate audio in the frequency domain or the time domain. For instance, in [7], a generative adversarial network is trained to generate music audio signals. The network is trained on the audio in the frequency domain rather than the waveform. The paper claims using the instantaneous frequency representation, the model outperforms other methods at generating coherent and more realistic sound. When assessing the quality of generated audio music, the evaluation focuses on the compositional quality, performance, and audio quality of the generated music. However, when assessing symbolically generated music, only the compositional aspects of the generated music are evaluated. My research interest is in the compositional level. Therefore, I am going to focus on symbolic music generation models only.

Since the early work of Arthur Samuel on machine learning there has been incremental progress toward more powerful models over the past six decades, in the past 10 years deep learning has been used to achieve impressive results in several applications. Deep learning is a branch of machine learning that uses artificial neural networks. In the earlier neural networks up until about 2010, it was common to have fewer than 10 layers to train the model within a reasonable amount of time. About 2010, with the use of Graphical Processing Units (GPUs) that are optimized for running a large number of computations in parallel and algorithmic improvements, it was possible to train networks with tens of layers. The term deep neural

networks was coined to refer to the networks with a large number of layers compared to the earlier neural network architectures that had about 10 layers at most. Almost all of the generative models discussed in this dissertation are based on deep learning.

A probabilistic view of the generative models provides a generic framework for all the models reviewed in this paper. In general, a set of random variables, $x_1, x_2, \ldots, x_n$ are sampled in a dataset. Estimating the joint probability distribution $P(x_1, x_2, \ldots, x_n)$ then allows us to sample new values. To draw new samples from the estimated distribution, the joint probability is used to estimate the marginal and conditional probabilities. The amount of computational resources needed for computing conditional probabilities is impractically large with the large n value required for some applications. For example, to generate small 100-pixel by 100-pixel images, n would be 10,000 which requires an impractical amount of computation.

A common approach to circumvent the mentioned problem with estimating the joint probability distribution is to represent the joint distribution of the variables as the product of conditional distributions. For instance, in [8], published in 2000, is an early research on using a neural network to learn the conditional distribution instead of the joint distribution.

Using the conditional probability equation, it is possible to replace the joint probability of two variables $P(x_1, x_2)$ with $P(x_1|x_2)P(x_2)$. Sampling this conditional probability is done by first sampling x_2 and then x_1 from the corresponding probability distributions. This is in effect an auto-regressive model. By repeatedly applying the conditional probability equation for n variables, the following result is obtained.

$$P(x_1, x_2, \ldots, x_n) = P(x_n|x_{n-1}, \ldots, x_1)P(x_{n-1}|x_{n-2}, \ldots, x_1)\ldots P(x_2|x_1)P(x_1), \tag{2.1}$$

which can be written in the more compact form

$$P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|x_{i-1:1}). \tag{2.2}$$

The conditional probability $P(x_n|x_{n-1},\ldots,x_1)$ still requires the same number of parameters as the joint probability. To reduce the complexity, the conditional probability is approximated by reducing the number of dependencies. This is typically done by setting an upper bound k on the number of dependent variables with $k < n$. Therefore, $P(x_n|x_{n-1},\ldots,x_1)$ is approximated by $P(x_n|x_{n-1},\ldots,x_{n-k})$.

In the example mentioned above with 10 random variables, using the conditional probability with $k = 2$, the model considers two previous notes to predict the next note. Given that each of the previous two notes can have 10 values, and the next note also has 10 possible values, to fully represent the conditional probability distribution 10^3 parameters are required. To simplify the calculations, assume that for each of the 10 notes, a conditional probability distribution needs to be estimated, requiring 10^3 parameters. Therefore at most $10,000$ (i.e., 10×10^3) parameters are needed which is much smaller than the 10 billion parameters of the joint probability distribution. Another common simplification is to use a single conditional distribution that is independent of the predicted variable's position within the sequence. Therefore, instead of 10 distributions, only one distribution is sampled to generate the 10 notes. Using an ML model, for example, a deep neural network, to estimate the conditional distribution has the advantage of generalizing for unseen data.

Section 2.3 reviews multiple music data representation methods and several publicly available music datasets specifically collected for training ML models. Depending on the end goal of a music generative model, some certain methods may be needed. That is why in Section 2.4 multiple applications of music generative models in the literature are reviewed. Several deep generative neural network architectures are common in the literature. Some of the salient architectures are reviewed in Section 2.5. Section 2.6 explains some algorithms that are used to train the generative models. Section 2.7 discusses some methods of evaluating the performance of music generative models. My critical view of machine learning methods is presented in Section 2.8. In Section 2.9, I subjectively evaluate some of the generated samples from my experiments on symbolic music generation with diffusion models. My concluding thoughts are

presented in Section 2.10.

2.3 Data

With deep learning models, the amount and quality of the training data are important factors in the quality of the data generated by a generative model. This is the reason considerable effort is spent by researchers on collecting training data. Data format and preprocessing methods can also impact a model's limitations and generalization capabilities. In this section, first, the common data representation for symbolic music for generative models is reviewed, and then some of the publicly available datasets that can be used to train generative models for symbolic music are briefly reviewed.

2.3.1 Data representation

Music data exists in either symbolic or audio representation. Symbolic music can be converted into audio using a synthesizer or real musical instruments, while raw audio data can be transcribed into symbolic music. Symbolic music can take on multiple possible representations, each with its own advantages. Reasons for preferring one representation over another include increased efficiency of the machine learning model or the convenience for the researcher.

Piano-roll

The earliest paper using piano-rolls to represent polyphonic music and training a generative model that I found in the literature is [9]. In this approach, the piano-roll is represented as a matrix. The rows are the pitches, and columns are the time ticks or time indices. Each column represents the shortest time duration chosen for the music. The elements with the value of zero in the matrix represent silence. Elements with the non-zero values indicate the presence of the corresponding pitch and time tick. Additionally, a non-zero element corresponds to the velocity in the MIDI format, which corresponds to the loudness. For example, if it is assumed that eighth notes are the shortest note duration, then each column in the piano roll matrix corresponds to

the time duration of an eighth note. In that case, a quarter note would be represented by two horizontally adjacent elements with some value between zero and one.

The paper [9] mentions transposing the music pieces in a common tonality; while not necessary, it helps the model training process. This transposition, in effect, reduces the variation in the training data and that in turn reduces the required complexity of the model. The downside is that a composer's style for certain keys is lost after transposition. It is possible to simplify the piano-roll representation by removing the dynamics. The simplified piano-roll is a binary matrix, encoding what pitches are present at what time without the velocity or loudness information. This simplification, in general, may turn learning the data for an ML model easier, at the cost of reducing the expressiveness of the music. An extra simplification step that is common is to also remove the duration of the notes and only keep the attack time of the notes. This simplification, similarly, makes training an ML model easier since there is less variation in the data.

The main downside of a piano-roll as a matrix is that the matrix is sparse. Therefore, a lot of computation is wasted since the model still has to process the zero values. Another approach to representing polyphonic piano-rolls is presented in [10]. In this approach, each instrument or MIDI channel is encoded separately. For instance, Bach Chorales are split into four separate channels, one for each voice. Splitting the piano-roll this way, allows restricting the number of present pitches at a time to exactly one. In [10], for Bach's chorales, each voice has only one note at a time. The benefits are twofold. First, the input data size is reduced since each MIDI channel is represented by a vector. The values in the vector represent the pitch at that time. Second, the model's output is simplified and easier to interpret. The output is a probability distribution over the pitches for each given channel per time index.

Sequence of events

Another approach to avoiding a sparse matrix input is using a sequence of events to represent the symbolic music presented in [11]. A sequence of integers represents the events. There are a total of 388 event types. 128 numbers are reserved to represent the MIDI pitch

note-on. 128 numbers are reserved to represent the MIDI pitch note-off. 100 numbers are reserved to represent the time-shift or advancing time in increments of 10 milliseconds, up to one second (100×0.01seconds). The velocities of the notes are quantized from 128 to 32 numbers. The advantage of this representation compared to piano-roll is that the data along the time axis is compressed and less wasteful. For example, a note held down for 16 ticks requires encoding the pitch over 16 time indices. In the sequence-of-events representation, only three events are used to encode the same held note for 16 ticks. Also, there is no need to use zeros to indicate silence. This input data size reduction in turn, may reduce the required model complexity.

2.3.2 Publicly available data

The Nintendo Entertainment System Music Database [12] contains 5278 songs. The authors also have published an open source synthesizer for this specific format. The songs can also be converted to MIDI. The large set of samples, along with only four available sound types, makes this a suitable dataset for researching machine learning-based methods for symbolic music generation, even though the generated music would have a narrow set of applications.

URMP (University of Rochester Multi-Modal Musical Performance) [13] is a dataset for facilitating audio-visual analysis of musical performances. The dataset comprises 44 simple multi-instrument musical pieces assembled from coordinated but separately recorded performances of individual tracks. For each piece, the dataset provides the musical score in MIDI format, the high-quality individual instrument audio recordings, and the videos of the assembled pieces.

GuitarSet [14] is a dataset of high-quality guitar recordings and rich annotations. It contains 360 excerpts, 30 seconds in length. The 360 excerpts are the result of 6 players, 2 versions (comping and soloing), 5 styles (Rock, Singer-Songwriter, Bossa Nova, Jazz, and Funk), 3 progressions (12 Bar Blues, Autumn Leaves, and Pachelbel's Canon), and 2 tempi (slow and fast).

The authors in [15] find YouTube videos of solo piano performances and use an open source automatic piano transcription model to generate 10,854 pieces in MIDI format. The large

size of this dataset allows training large ML models. One weakness of this dataset is that some of the pieces are not transcribed correctly in this dataset. Another problem is that the name of the piece and composer are extracted from the title and description of the YouTube video which in some cases are incorrect. Therefore, filtering based on composer name might not be completely accurate.

The MAESTRO [16] dataset contains about 200 hours of paired audio and MIDI recordings from ten years of International Piano-e-Competition. The MIDI data includes key strike velocities and sustain, sostenuto, and una corda pedal positions. Audio and MIDI files are aligned with an accuracy of about 3 ms and sliced to individual musical pieces, which are annotated with composer, title, and year of performance. This is one of the most commonly used datasets in the literature for training music generative models.

The Groove MIDI Dataset (GMD) [17] is composed of 13.6 hours of aligned MIDI and (synthesized) audio of human-performed, tempo-aligned expressive drumming. The dataset contains 1,150 MIDI files and over 22,000 measures of drumming. An expanded version of this dataset with 444 hours of drum recordings is presented in [18]. This is another commonly used dataset for training symbolic music generative models.

POP909 [19] is a dataset that is specifically created for training music arrangement generation models. It contains multiple versions of the piano arrangements of 909 popular songs created by professional musicians. The MIDI file of each of the songs contains the vocal melody, the lead instrument melody, and the piano accompaniment.

2.4 Applications

The first step in developing a generative model is deciding on the problem that the model is intended to solve. A single model could possibly be designed to be used for multiple applications. This section discusses some of the common use cases of music generative models, both symbolic and audio.

2.4.1 Music transcription

Generative models can be used to assist automatic polyphonic music transcription methods [9]. In this application, the generative model is used to predict probability distribution of notes in the next measure based on the previous measures. For this purpose, the prediction accuracy only at a small temporal scale is critical to the performance of the complete transcription method. The predicted probability distributions over the notes generated by the (symbolic) generative model are generated and, independently, the audio-only transcription model generates the note probability distributions. Once both sets of note probability distributions from the generative model and the audio-only transcription model are ready, then they are fused together to generate the output of the combined transcription model which is the transcribed score of the audio. More specifically Equation 2.3 computes a cost function and the notes that minimize this cost are predicted to be present notes in the audio input:

$$C = -\log P_a(v^{(t)}) - \alpha \log P_s(v^{(t)}|\tilde{A}^{(t)}) \tag{2.3}$$

where $\tilde{A}^{(t)}$ is the transcribed notes up to the current time, $v^{(t)}$ is the input audio frame at time t, $P_a(v^{(t)})$ is the probability distribution from the audio transcription model, $P_s(v^{(t)}|\tilde{A}^{(t)})$ is the probability distribution from the symbolic generative model, and α is the weight of the generative model in the cost value. In this method, the seven most likely notes from the transcription model are taken and all combinations of those seven notes are generated. For each of those combinations, Equation 2.3 is evaluated, and the combination with the least cost value is selected as the output of the transcription method. If the generative model is trusted, the weight α is set to a high value, for example, 1; otherwise, it is set to a lower value, for example, 0.1.

2.4.2 Co-composition

Generative models can also be used for co-composition. Co-composition refers to the interaction of a composer with a generative model to compose music. One motivation for such

an application is to help with inspiring the composer with generated music or reduce the amount of effort needed to write new music. There are several possible ways a model can be developed to allow interaction with the composer. In this section, some of the different approaches to co-composition are reviewed.

The approach presented in [10] for co-composition is based on editing a score in the piano-roll form. The user is given a blank four-measure piano-roll and can place eight notes or leave the measures partially or completely blank. The model, named Coconet, tries to improve and complete the given piano-roll from the user. In section 4, the model is discussed in more details. This is a low-level control over the generated symbolic music. If the generated music is not desirable to the user, then the user can modify the score again and reiterate. The user does not have direct control over the generated motifs, for example. Therefore, Coconet falls under a class of music generation methods that are single-step and also called end-to-end methods. The end-to-end methods take some input from the user and in one step generate the music. There is no intermediate representation and control in such methods. Most of the music generative models follow the end-to-end approach. This includes the recurrent network in [11], which is reviewed later in this chapter, and generates a sequence of MIDI events in a single step.

An interface for novices to interact with the Coconet model [10] is presented in [20]. The main idea of the interface is to reduce the cognitive load of the user. This is done, for instance, by showing multiple sampled outputs that the user can choose one from. The paper also uses soft priors to let the user select pitches that they would prefer. The model then tries to satisfy those requests while following the style of Bach. The pitches provided by the user are called soft priors because they may also be overwritten with some other pitches by the model.

The multi-step generation methods compared to end-to-end methods have intermediate steps and music representations. For example, a system may first generate the rhythm for the piece. The generated rhythm then can be inspected and modified by a musician. Another model then takes the rhythm and generates the melody. This approach may appeal to some musicians or composers as it offers more artistic control over the final generated music. The music generation

method proposed in [21] is a multi-step method. The argument in favor of multi-step methods presented in [21] is two-fold. First, the models for each step are simpler. Second, the simpler models require less training data. The same models can be used in both one-step and multi-step music generation methods.

Style transfer is another approach to co-composition. In general, a music style transfer model takes a piece of music in one style and generates a piece of music that strongly resembles the original pieces but in another style. The style transfer models were made possible by generative adversarial networks. For example, the GAN model is trained on two styles of music, classical and jazz. It is also trained to take a jazz piece to classical style, and then back again to jazz. At the end of this process, the final transformed piece should be identical to the original, and any differences between them are used as errors to train the model. Therefore, the model does not learn to generate new music but transfers the style of a given piece. The method proposed in [22] based on this idea trains a model to convert music between jazz, classic, and pop. Authors in [23] propose to use a commercial product, Band-in-a-Box, to generate arrangements of the same music in different styles. Since this provides directly a music and its equivalent in another style, it is possible to train a model similar to language translation models for this purpose. The paper uses recurrent neural networks with attention modules (discussed in section 4) that first encodes the input music in the latent space and then decodes from the latent space to another style.

A related application is using a generative model to improvise with, interactively. Dubnov et al. in [24] propose an approach to use a generative model to improvise with the input music it receives.

2.4.3 Music production

ML algorithms are also used in tools specifically designed for music production. For instance, designing drum sounds is a task in music production. In [25], the authors propose a new method for generating drum sounds using variational autoencoders. The autoencoder is conditioned on some parameters that the user controls to get the desired drum sound. The

authors in [26] attempt to evaluate, in practice, the use of ML models in music production. Their conclusion is that there is plenty of room for improvement in the methods and models and close collaboration with music producers and artists is needed to develop practical tools.

2.4.4 Synthetic Instrument

With the advancement in developing sequential models with large inputs, it has been possible to use deep networks to generate audio directly in the time domain. The motivation in [27] is to synthesize music scores using deep networks to have a more expressive performance than common synthesizers. Using the model it is possible to morph between the instruments and create sounds that are with varying degrees between two instruments, for example trumpet and violin.

2.5 Models and network architectures

This section reviews the deep neural network architectures that are used in the literature review for music generation.

2.5.1 Feedforward networks

The simplest network architecture is feedforward neural networks. In a feedforward network, the neurons of each layer are connected only to the neurons of the previous layer. Many of the feedforward networks in the literature use convolution layers. Convolutional neural networks [28] reduce the number of parameters (which are the weights) using parameter sharing in filters and can more easily learn spatial relationships in the data than a fully connected network. There are some variants of feedforward architecture, for instance residual networks [29], that contain layers that are connected not only to the previous layer but also to the layer before the previous layer.

In [10] a residual feedforward network, named Coconet, is trained to generate music in the style of Bach's chorales. The paper also proposes an iterative process to sample the

generative model. The main feature of this method is that with partial scores as input the method generates complete scores in the style of the training set. Their approach is based on order-less auto-regressive models [30]. The following example illustrates the reason for calling the model order-less. In the introduction section, it was discussed how an autoregressive model is based on a sampling order. This example,

$$P(a,b,c) = P(a|b,c)P(b|c)P(c), \tag{2.4}$$

shows a particular sampling order was chosen to replace the joint probability with conditional probabilities. Given in the application Coconet is intended for, any of the variables (notes) could be provided by the user and the rest are left for the model to generate. For example, the user may provide only the value for variable b (the middle note). If the neural network has learned only the particular conditional order, then it would have to first sample c, independent of the value of b. Thus there is a chance, the probability $P(b|c)$ for the given values of b (given by the user) and c (sampled by the method independent of b) may be zero or near zero. Therefore, the model would fail to fill the missing notes in the piano-roll in the style of Bach's chorales.

The orderless sampling method addresses the problem with a model that has learned a particular conditional order, by training the model to learn all the possible conditional orders. This means for three variables, like in the example above, the model should see training samples from all possible combinations of notes being absent or present to learn these probability distribution: $P(a)$, $P(b)$, $P(c)$, $P(a|b)$, $P(a|c)$, $P(b|a)$, $P(b|c)$, $P(c|a)$, $P(c|b)$, $P(a|b,c)$, $P(b|a,c)$, and $P(c|a,b)$. Thus, in the previous example, the note value for b is provided and the model samples $P(a|b)$ and $P(c|b)$. This still has the problem that it is assuming a and c are independent always. To address this issue, the iterative process in Coconet, after the first step, randomly masks some notes and samples their value using the model. For example, if a is masked, then the probability $P(a|b,c)$ is used to sample a value for a. At the end, the Coconet method has followed this sampling order $P(a,b,c) = P(a|b,c)P(c|b)P(b)$. With this order, the dependencies

between the variables are not ignored.

Another problem that Coconet might face is that the user may provide unlikely notes or combinations of notes. In the example discussed above, the user's note for b could be an unlikely note considering $P(b)$. In that case, regardless of the sampling order the generated sequence of notes are not going to follow Bach's style. For this reason, the iterative process that Coconet uses to mask some notes and re-sample them includes the notes provided by the user.

The iterative process described above that masks some notes and then resamples them using the estimated probability distributions is called annealing blocking Gibbs sampling. It is called blocking because adjacent notes on the time axis are masked together. For example, if the width of the block is 10 time ticks, all the 10 adjacent notes at a random place are masked. It is also called annealing because the width of the blocks used to mask the score are at the beginning of the process large and over each iteration the block width are reduced. Fig. 2.1 illustrates the Gibbs sampling process by Coconet on an example.

In Coconet, the input piano-roll is separated into four separate piano-rolls, one for each voice. The method also simplifies the problem by assuming that at every time index in the piano-rolls there is one note present, that is in the combined four voices there are four notes present at each time index. Therefore, Coconet does not generate any rests or silences in the score. In the following $\mathrm{x}_{i,t,p}$ denotes piano-roll elements from the training data for voice i, time tick t, and pitch p. Each $\mathrm{x}_{i,t,p}$ is a binary value indicating whether pitch p is present in voice i and time t.

The neural network needs to learn during the training phase all possible conditional probabilities using the following process. The network is designed to predict the probability of the pitch of the notes in the output. The softmax function is used to convert the outputs of the last layer to probabilities:

$$p_\theta(\mathrm{x}_{i,t,p}|\mathrm{x}_C) = \frac{\exp(h^L_{i,t,p})}{\Sigma_p \exp(h^L_{i,t,p})} \tag{2.5}$$

where p_θ is the probability function with the parameters Θ, which are the weights of the neural

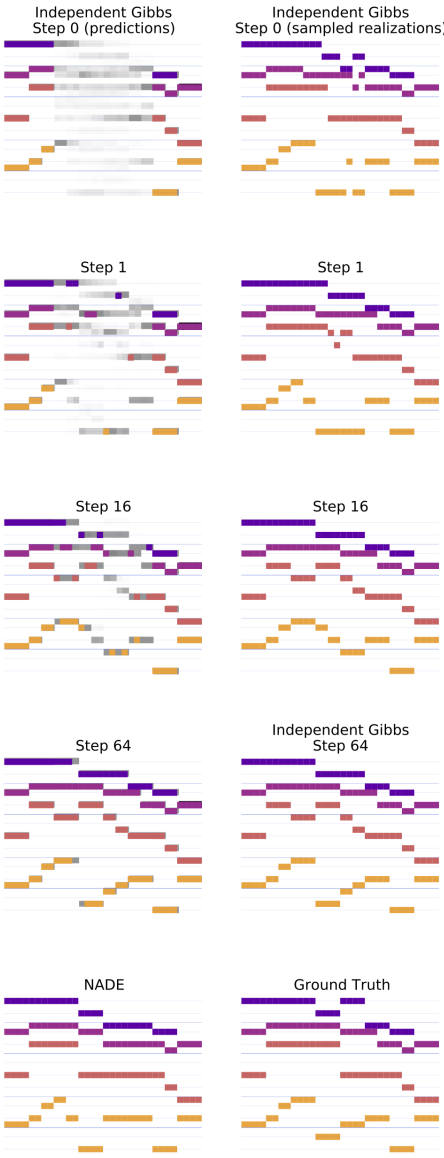


Figure 2.1. The steps of Gibbs sampling done by Coconet. Each of the four colors corresponds to one of the four voices in the Bach Chorales. In the left column, the masked notes are replaced by the probabilities for notes predicted by the neural network. The darker gray indicates a high probability, i.e., closer to one, and lighter gray indicates a low probability, i.e., closer to zero. The right column shows the filled score by sampling the probability distributions from the left column. Source: Reproduced from [10], licensed under CC BY 4.0.

network, $h^L_{i,t,p}$ is the output of the last layer, labeled L, for voice i, time tick t, and pitch p. For each training sample, x, which is a 4 voices piano roll matrices, is masked randomly. x_C is the set of notes in the masked piano roll that are not masked. The estimated probability distribution

allows adding some randomness to the generated music by sampling pitches other than the one with the highest probability.

The Adam optimizer (discussed in Section 2.6.1) and the cross-entropy loss function are used to train the classification network. More specifically the loss function is defined as

$$L(\mathrm{X};\mathrm{X}_C,\theta) = -\sum_{(i,t)\notin C}\sum_{p}\mathrm{X}_{i,t,p}\log p_\theta(\mathrm{X}_{i,t,p}|\mathrm{X}_C) \tag{2.6}$$

where the logarithm of the predicted probability is multiplied by the binary value whether that pitch is present for voice i and at time t in the input piano-roll, and the result is summed for each pitch p of the masked notes. The loss is summed for each voice and time (i,t) that are masked in the input piano-roll, $(i,j) \notin C$. In essence, this loss function returns a large loss value when the note at time t is supposed to have the pitch p but the predicted probability for that pitch by the model is smaller than 1.0. In Section 2.6.1, the cross-entropy loss is discussed further.

When generating new music, the user passes an incomplete score for the four voices. The model then generates a probability distribution for each note, and samples the distributions to complete the score. In the completed score with a pitch assigned to every note, a randomly selected subset of notes are masked. The new masked score goes through the same process of completing the score and masking notes several times. These iterations are part of the annealing blocked Gibbs sampling process. In each iteration fewer variables are sampled. The generated score at the end might not have preserved the notes entered by the user as the sampling process might replace them.

2.5.2 Recurrent networks

Recurrent networks, unlike feedforward networks, retain memory of the previous input data inside the network by feeding the output from some layers back into the network. But the network's memory is limited and imprecise. In [11] a recurrent neural network (RNN) that is trained on a piano MIDI dataset. The model is trained on the input data that is converted into the

sequence of events format, which is probably an optimal representation for RNNs. The presented model is a regular RNN without any modification. RNNs are less frequently used in the literature after the use of other architectures such as transformers and diffusion models increased.

2.5.3 Generative adversarial networks

Generative adversarial networks (GANs) are composed of two models that compete against each other [31]. The generative model is given a noise vector and it generates the output that is then fed to the second model. The second model is a classifier, called the discriminator, that is tasked with telling apart generated and real inputs.

The generator does not get to see the real data, and it learns by getting the error backpropagated from the discriminator. The discriminator is provided with the mixed samples from the generator with real training samples, and should tell which one is real. As the training process progresses, the discriminator gets better at classifying the input as real or generated. And the generator gets better at generating outputs that can fool the discriminator by being more similar to real training samples. At the end of the training process, the discriminator fails most of the time at telling apart the real and generated samples as the generator has learned to generate samples that seem to come from the distribution that the real samples are drawn from.

GANs are known to be difficult to train. Because there are two networks competing against each other, there is a chance one improves its performance more than the other to the point that the other network cannot improve any further. Some variants of GANs try to mitigate this issue. Another issue with GANs is that they are unlikely to be able to generate outputs as diverse as their training data. That means for example, if 100 music scores are used to train a GAN, it will not be able to generate any music score that is similar to some of those 100 scores. Variational autoencoders and diffusion models are claimed to be able to generate output as diverse as the training data.

2.5.4 Autoencoders

Autoencoders are models with an hourglass architecture. The name of the architecture refers to having the layer with the smallest number of neurons at the middle of the neural network. This middle layer is also called the bottleneck layer. The first half of the layers from the input to and including the bottleneck layer is considered the encoder network as it transfers data to a lower dimensional space at the bottleneck layer. The output space of the bottleneck layer is also referred to as the latent space of the autoencoder. The second half of the network is considered the decoder network which takes the low-dimensional representation of the data points and reconstructs the original input. In autoencoders, the reconstruction error which is commonly the Euclidean distance between the input and the output is minimized. The distribution of the training data points in the latent space is unconstrained, and tend to be unstructured.

Variational autoencoders (VAEs) [32] are models that have a loss function composed of two terms. The first term measures the reconstruction error, similar to autoencoders. The second term is a regularizer that forces the distribution of the training data points in the latent space be compact and centered around the origin. This is beneficial to the generative models as it is possible to pick a random point in the latent space, and the decoder will generate an output that appears to be drawn from the same distribution of the training data points.

MusicVAE [33], proposed in 2020, uses variational autoencoders to generate symbolic music. Because the VAE learns the joint probability of the input variables, it is computationally prohibitive to train the VAE model on long sequences. In MusicVAE, the VAE is trained to generate only two measures. To generate longer sequences the VAE is combined with recurrent neural networks (RNN) to generate eight measures with coherent melody. This approach, using RNNs to generate longer sequences of music, appears to be abandoned in the literature in favor of transformers, which is discussed in the next section.

2.5.5 Transformers

The use of transformer models to generate music was introduced in [34]. All the generated pieces by this method are about 60 seconds. This is because the model shows consistency and structure only up to that scale. The authors state that at a larger scale, the generated pieces feel lost without any high-level structure. By listening to the samples, it is obvious the model does not follow a certain structure at the high-level. This method is based on a method used for natural language processing and translation. Many neural network architectures that deal with sequential data were originally designed for language processing and translation. Transformers address the shortcomings of recurrent neural networks.

In order to apply transformers to symbolic music, the authors in [34] use the sequence of events representation that was discussed in section 2. A token is the smallest unit of data that can be separated from the rest of the sequence. In language processing each word is considered a token, for example. A token in this method is an event in the music data. For instance, there is a token for the onset time of each note and the note durations. The transformer model can look back at the generated notes (tokens) unlike RNNs in one single pass without iterating over the past tokens. The network in [34] can look at a maximum 2,000 tokens in the past. That is on average about 60 seconds with a solo piano piece.

An example using the sequence of events: <BOS> A2-begin, progress-time-10ms, A2-stop, C2-being, E2-begin, progress-time-20ms, C2-stop, E2-stop <EOS>. <BOS> means beginning of score and <EOS> means end of score. A2-begin and A2-stop represent the onset and release time of the note. The token progress-time-10ms indicates 10 milliseconds has passed since the previous event.

That means the model looks at the past 60 seconds to decide what the next note should be, how long that note should be held, and the dynamic of the note. This allows the model to generate for example multiple variations of a given motif. The previous methods could not generate such patterns of repetitions and variations as well as this model. The simple way to let a

neural network know about the past is to increase the input size of the network. For example, to let the network see 2000 previous tokens, the input size needs to be 2000. This makes the network very large and the training and inference will be very slow.

Recurrent Neural Networks (RNNs) are the most common approach to remember the past tokens. RNNs process the tokens and keep a hidden state vector that summarizes what they have seen in the past. But the problem is that RNNs forget the earlier tokens after seeing about 100 tokens. The attention method, which is used by transformers, is designed to let the network see a large amount of past data without being very slow. That means the model does not forget the earlier data as it has direct access to all the past tokens in the sequence. Attention is more efficient than simpler methods but it is still not very efficient. Once the input size is about 2000 tokens it becomes too slow to train it and do inference with. That's the reason the method in this paper does not look at the tokens in the past 5 minutes and looks only at the tokens in the past 1 minute.

A new variation of the sequence of events is proposed in [35] to encode MIDI files. The encoded music data is used to train a transformer model similar to the paper discussed above [34], which then can generate new music. Some generated samples by this method can be found on this page. They first argue piano rolls are inefficient for representing music data. They mention the "sequence of events" representation (encoding) is more efficient. They criticize the absolute times used in that approach. They suggest the time in the presentation should be based on beats and independent of tempo. This way, the same phrase but at two different tempi are identical to the model, which makes the learning process easier for the model.

An example using their proposed encoding looks like this: <BOS>, tempo:80, piano, velocity:72, A2:24, E3:24, C4:12, wait:12, D4:12, wait:12, C3:24, G3:24, E4:24, <EOS>. temp:80 sets the tempo to 80 bpm. Having a token for tempo allows the model to learn to set and change the tempo. velocity:72 set the MIDI note velocity. A2:24 represents playing note A2 for $24 \times 1/12$ beats (a constant resolution the authors chose), that is 2 beats. wait:12 means the notes after this token are $12 \times 1/12 = 1$ beat later start playing. The model is given 1024 tokens

(which is equivalent to almost 400 beats with their encoding method) and is asked to predict the next token (event). The predicted token is added to the end of the previous 1024 tokens. Then again, the last 1024 tokens are used to predict the next token. The model is trained on 3500 rock music MIDI files and 12,000 classical music pieces.

2.5.6 Diffusion

A method to generate symbolic music using diffusion models is proposed in [36]. In this method, a pre-trained VAE [33] is first used to convert two bars of music into a latent vector. The latent vectors of every music score then goes through the diffusion process. To sample new music scores, noise is converted into latent vectors by the diffusion model, which is then converted to piano-rolls by the decoder of the VAE.

Let $q(x_0)$ be the probability distribution the training data is sampled from, $x_0 \sim q(x_0)$. The forward diffusion process $q(x_t|x_{t-1})$ which adds Gaussian noise at each time step t, according to a known variance schedule $0 < \beta_1 < \beta_2 < \cdots < \beta_T < 1$, is defined as

$$q(x_t|x_{t-1}) = N(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I), \tag{2.7}$$

where I is the identity matrix. That is at each time step t, more Gaussian noise is added to the input data with the mean of $\sqrt{1-\beta_t}x_{t-1}$ and variance of β_t.

Diffusion models using a neural network approximate the conditional distribution $p(x_{t-1}|x_t)$, which is the inverse of q and is also a Gaussian distribution. By sampling some random Gaussian noise X_T and gradually denoising the noise sample X_T, a sample from the real distribution x_0 is eventually obtained. Specifically, the neural network attempts to learn the mean of the Gaussian distribution with t as the parameter and assume the variance is constant.

The authors in [37] note that the combination of q and p can be seen as a variational auto-encoder. Hence, the variational lower bound can be used to minimize the negative log-likelihood with respect to ground truth data sample x_0. It follows that the variational lower bound for this

process is a sum of losses at each time step t, $L = L_0 + L_1 + \cdots + L_T$. Consequently, each term of the loss becomes the KL divergence between the two Gaussian distributions, which can be written explicitly as an L2-loss with respect to the two means.

It is possible to directly sample x_t at any arbitrary noise level without the iterative forward process. This is done by the equation

$$q(x_t|x_0) = N(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) \tag{2.8}$$

where $\alpha = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$. This means a random t can be selected and the corresponding term of the loss function L_t can be computed directly and optimized during the training phase.

After some simplification it is shown that the neural network can estimate the amount of Gaussian noise and not just the mean of the Gaussian distribution. Therefore, the loss function that is minimized is

$$\|\varepsilon - \varepsilon_\theta(x_t, t)\|^2 = \|\varepsilon - \varepsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{(1-\bar{\alpha})}\varepsilon, t)\|^2, \tag{2.9}$$

where ε is the noise amount added to the input at step t, $\varepsilon_\theta(x_t, t)$ is the estimated noise amount by the neural network, x_t is the input sample with noise at step t.

2.6 Training methods

While the neural network architecture and training data are important elements for training generative models, another equally important factor is the loss function and the optimization method to optimize the networks and minimize the loss during training. This section also summarizes the different ordering strategies used by autoregressive methods.

2.6.1 Optimization

One of the challenges for training generative models is designing a loss function or functions that are then minimized using a variant of gradient descent based optimization method. Using gradient descent to minimize the error requires computing the gradient of the loss function L with respect to the weights θ of the model for all the data points in the training set, picking a gradient descent step size η, which is called the learning rate, and update the weights according to the equation

$$\theta = \theta - \eta \cdot \nabla_\theta L. \tag{2.10}$$

When the whole data set is used to compute the gradient, the method is called batch gradient descent. Mini-batch gradient descent, which uses a small number of training points to compute the gradient, is almost always preferred to batch gradient descent. The reason is that with the current hardware, computing a mini-batch gradient for the whole dataset is faster than computing the gradient of the whole set. Given that mini-batch gradient descent is also to some degree stochastic, it is less likely to get stuck in local minima.

Picking a value for the learning rate η in practice is not easy. A small value would slow down the training process. A large value could make the gradient jump around randomly without converging. The optimal value of the learning rate depends on many factors, including the architecture and the training data. Therefore, the optimal learning rate value in a particular setup might be too large or too small for another. Finding the optimal value with trial and error is slow and frustrating for researchers. Adaptive Moment Estimation, often referred to by Adam [38] is a variant of gradient descent that is less sensitive to the learning rate parameter. The main advantage of Adam contributing to its popularity is auto-scaling the learning rate based on the previous gradients. Methods like Adam do not require precise tuning of the learning rate parameter. This is the reason why Adam is widely adopted by researchers in the literature.

Adam estimates the mean and variance of the gradients using the decaying averages:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla L_t \tag{2.11}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla^2 L_t, \tag{2.12}$$

where m_t and v_t are the estimated mean and variance at the time step t, β_1 and β_2 are the parameters, and $\nabla^2 L_t$ is the squared gradient value of the loss function at time step t. The authors propose default values of 0.9 for β_1 and 0.999 for β_2. Therefore, m_t is maximized when all the gradients are in the same direction. m_t can get close to zero if the gradients are in opposite directions. v_t is independent of the direction of the gradients, and depends only on the magnitude of the gradients. In the following, it is shown how m_t and v_t are used together to adjust the learning rate. The initial values m_0 and v_0 are set to zero. These initial values therefore bias the estimated mean and variance for the first few time steps to a low value. The authors use the following equations to define new variables $\hat{m}_t$ and $\hat{v}_t$, which are less affected by the initial value bias.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{2.13}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}, \tag{2.14}$$

where β_1^t and β_2^t are β_1 and β_2 to the power of t.

These bias corrected values are then used to update the weights,

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\varepsilon}\hat{m}_t. \tag{2.15}$$

Compared to gradient descent in Equation 10, the learning rate η is multiplied by the ratio of $\hat{m}_t$ over $\sqrt{\hat{v}_t}$. This ratio is a large value, if the magnitude of the gradients are large and gradients are in the same direction. In such cases, the optimization can converge quicker, if the learning rate

is increased. And this ratio is a small value, if the magnitude of the gradients are small or the gradients are in different directions. In such cases, the optimization is more likely to converge when the learning rate is reduced. Therefore, ADAM reduces the sensitivity of gradient descent to the values of the learning rate parameter. The authors propose default value of 10^{-8} for ε to avoid very large update steps when $\sqrt{\hat{v}_t}$ becomes a small value.

Coconet [10], a feed forward network reviewed in this paper, uses ADAM to optimize the parameters of the network and the loss function is the weighted cross-entropy. A sample from the training data is drawn and some of the pitches are masked. The loss function, cross-entropy, is weighted by the number of masked pitches as it is claimed to ensure consistent estimate of the joint negative log-likelihood. The network weights are optimized using Adam. The cross-entropy, H, between a true distribution q and an estimated distribution p is defined as

$$H(p,q) = -\sum_{x} q(x) \log p(x). \tag{2.16}$$

Coconet uses the softmax function to estimate the probability of each pitch at each time tick. The true distribution has all the probability mass on the correct pitch, for example $q = [0,\ldots,1,\ldots,0]$. Therefore the cross-entropy loss that Coconet uses for a single note $\mathrm{x}_{i,t}$ in the piano-roll has the form

$$L(\mathrm{x}_{i,t}; \mathrm{x}_C, \Theta) = -\log\left(\frac{\exp(h^L_{i,t,p^*})}{\sum_p \exp\left(h^L_{i,t,p}\right)}\right), \tag{2.17}$$

where $h^L_{i,t,p}$ is the output of the last layer for pitch p and h^L_{i,t,p^*} is the output of the last layer at the index that corresponds to the correct pitch from the training data. the above equation is equivalent to

$$L(\mathrm{x}_{i,t}; \mathrm{x}_C, \Theta) = -h^L_{i,t,p^*} + \log \sum_p \exp(h^L_{i,t,p}), \tag{2.18}$$

which is also commonly used in literature.

2.6.2 Autoregressive order

For chronological data, an autoregressive generative model seems to be a natural fit. But if the data is generated in an offline fashion, such as in a music score that is generated and then presented in its completed state to the user, the chronological order is no different from any other order. To train an autoregressive model, the sampling order at inference time should be decided. The reason is that the model must learn conditional probability distributions that are used during the generation phase. Therefore, the straightforward approach is to pick the chronological order as the order of the autoregressive model.

There are in general three types of autoregressive models based on the sampling order. First, unidirectional models that can generate tokens in a certain order, for example a model that generates text from left to right. Second, bi-directional models are trained such that they can generate in one of the two orders. Third, omnidirectional models that are trained in such a way that any order of sampling is possible. The Coconet model that was discussed in the previous section falls under the omnidirectional category. The downside of that method is that a slow iterative Gibbs sampling process is required.

It is possible to look at diffusion models through this lens. In this view, diffusion models learn the joint probability distribution and iteratively adjust the sampled values to be more likely drawn from the training data distribution. Therefore, these models could be used to generate symbolic music with the dependency between all the notes considered in the score. As a result, exploring the use of diffusion models to generate symbolic music seems to be an exciting research project.

2.7 Evaluation approaches

The evaluation methods used in the literature for the generative models are divided into subjective and objective metrics. The objective metrics, while not perfect, simplify comparison of methods. The subjective metrics are more difficult to collect but may be more informative

than objective metrics.

A commonly used evaluation method for image generators is the Fréchet Inception Distance (FID) [39]. In FID, a classifier that has been trained on real images is used to convert the generated images and natural images to a latent space. This is done by passing the images to the classifier and taking the output of the layer before the last. The idea is that the layer before the last layer contains the information necessary to classify the image. Therefore, the neural network up to and including the layer before the last can be used as a dimensionality reduction method. After getting the latent space representations of the two sets of images, real and generated, a multivariate Gaussian distribution is fitted on each set of latent vectors. The comparison of these two Gaussian distributions is expected to inform about the quality and diversity of the generated images compared to the natural images.

The authors in [40] mention three downsides with FID and other similar methods like maximum mean discrepancy [41]. First, a pre-trained classifier is needed which is not available for a wide range of applications. Second, the assumption of the distribution of the data being Gaussian almost certainly does not hold. Third, using a single metric for both diversity and quality is difficult to interpret. To overcome these shortcomings, the authors in [40] propose two new metrics, density and coverage. In this method, instead of using a pre-trained classifier to reduce the dimensionality of the data, a deep neural network with random weights is used. Such a randomly initialized network without training is shown to generalize well for multiple applications. This network is used to convert the natural and generated images to latent vectors. In the latent space, a hypersphere is fitted on each natural image vector with the radius being the distance to the nearest natural image vector. Density is defined as the average of the hyperspheres that contain each of the generated latent vector. Therefore, a high density means the generated samples are similar to the training samples. Coverage measures the fraction of the training samples that have at least one generated sample in their hyperspheres. A high coverage value of one means the generated samples have a similar amount of variations and differences as the training samples. A low coverage value close to zero means that the generated samples are

concentrated in a small area of the sample space. For example, for a model that is trained on a large set of music scores, if it generates music that is mostly similar to each other, the coverage score will be a small value.

Some subjective metrics are used in [7] to evaluate the performance of the generative model. For instance, a pair of audio examples are played for the participants and the participants are asked to decide which sound they prefer. Then the authors count the number of times each model is preferred. There is a clear margin in this preference metric between the evaluated models.

2.8 Beyond statistical models

In this section, I discuss my critical view of machine learning methods. In particular, I go over what is the strengths and weaknesses of the machine learning for music generation. At the end, I explain my opinion on the future research in the area of machine learning-based symbolic music generation.

While the methods reviewed in this paper are, to a limited extent, applied to generate music, the music produced does not fully reveal the strengths and weaknesses of these machine learning approaches. This is because numerous factors—such as the genre of the training data, the dataset size, and the computational resources dedicated to model training—directly influence the output. As a result, it is challenging to draw general conclusions about the pros and cons of these methods based solely on the limited music generated. In contrast, far more resources have been allocated to generative models for images and text, driven by their wider range of commercial applications. Given that my goal in this review is to inform my research on machine learning-based symbolic music generation, I focus on broader questions, such as the merits of pursuing machine learning methods for music creation and whether these systems can assist composers.

The main criticism of generative machine learning methods is that they are nothing more

than a statistical model. Critics use the term statistical model to imply these generative models are not capable of reasoning. And these models just learn the common patterns in the training data. For example, a generative model can generate symbolic music but does not have the means to predict how people would perceive that piece of music. In the past 10 years, the machine learning models have become larger, i.e. with orders of magnitude more parameters than the models of a decade ago. Often the term large models is used to signify this development. And regardless of the model size and amount of training data, a machine learning model is exposed to a narrow view of the world through the training data. Therefore, the main criticism holds for large models.

Generative machine learning models lack the facility for self-criticism and refining their output. This facility is often claimed to be a requirement for creativity. For instance, Agüera y Arcas [42] argues a person to write a long coherent text needs inner dialogue, deliberation, and iteration. And a machine learning model requires the same abilities to generate a long coherent text, such as a 10 pages article. Therefore, given large machine learning model lack these abilities, the question that remains is whether working on generative machine learning for music generation is futile.

For three reasons, I believe, despite the above criticism, it is worthwhile to work on generative machine learning models. First, tools that assist artists do not necessarily require generative models that criticize and improve the produced artifacts on their own. The reason is that in such a setting, the artist interacts with the system, interprets and evaluates the system's output, and iteratively refines the input to the system until a satisfactory output is generated. The artist also does not need to take the produced artifact as the final product and very well can modify it using other tools or inspired by that produce a completely new work of art. The term human-in-the-loop could be borrowed to describe the system including the user to deliberate and refine the generated artifact.

Second, generative machine learning models are getting better at learning patterns from data. Large models may turn out to possess at least some degree of ability to critique. The chat

system that uses GPT, which is a large language model, generates 20 responses to each prompt from the user. Then the same language model evaluates the 20 responses to determine which of them is not offensive [42]. The first non-offensive response there is displayed to the user. Extrapolating from this example, it is possible that larger models will be able to evaluate aspects of their own output. For example, a music generative model may also be used to evaluate the coherence of a generated music at the composition level or style.

Third, large machine learning models demonstrate surprising emergent abilities. For example, natural language models with 100 million and fewer than 13 billion parameters almost always give a wrong answer if asked to add two multi-digit numbers [43]. The surprise is that the models with 13 billion parameters and more are substantially better at answering the addition questions. Another example of emergent abilities is solving multi-step math word problems that only models larger than a certain size are able to solve. An example of such math word problems is presented in [43], "Tom's ship can travel at 10 miles per hour. He is sailing from 1 to 4 PM. He then travels back at a rate of 6 mph. How long does it take him to get back?"

The evidence for emergent abilities suggests that the machine learning model should not be hastily dismissed as just a statistical model that lacks reasoning and deliberate consideration. Therefore, researching machine learning models for music generation may also reveal some emergent abilities and creativity. Developing a better understanding of the limitations of these models is one of my research goals.

A barrier to larger generative models is the limited availability of training data. A larger model requires more training data. This means the size of models may eventually be constrained by the amount of available training data. This limitation is also evident in data subsets with high variability and limited training samples. For example, hand images in the training dataset exhibit high variability, often due to occlusion. In practice, a training dataset with an impractical large number of samples is required for a generative model to learn the distribution of hand images. As a result, generative models generally struggle to generate plausible hand images. In Fig. 2.2, sample images illustrate how generative models struggle to produce hand images

Figure 2.2. Hand images generated using Stable Diffusion. Image generative models are likely to generate distorted hands in the images. As a small fraction of the training images for these models have hands visible, the model has not seen enough hand samples. Midjourney, Stable Diffusion, and Dall-E are known to struggle with generating images containing hands.

without obvious distortions. A more in-depth discussion of this subject is presented in Chapter 4.

2.9 Experiments with diffusion

In this section, I present some high-level approaches to controlling and directing the music generated by diffusion models. This high-level control is readily available as diffusion models can be conditioned on a part of the piano-roll. Therefore, with a proper interface, an end user can use a diffusion model for several musical applications such as harmonization of a melody and completing a piece. However, in this section, I give the diffusion model complete freedom to compose new pieces.

Using the diffusion model I trained, I generated several pieces and a selected number are presented in this section. The ratio of present pitches (ones) to silents (zeros) in the piano-roll is a parameter in drawing a sample from the binomial distribution, which is explained in the next

chapter. In my experiments it appears that the effect of this ratio is stochastic and has no obvious effect on the generated piano-rolls. In some of my experiments, I experimented with different ratio values. For example 0.07, which is close to the average ratio of ones to zeros in the training piano-rolls.

My diffusion model uses a convolutional neural network that is able to generate a piano-roll of any length, even though it was trained on 16-beat long piano-roll segments. Therefore, in the generated piano-rolls, it is not expected beyond 16 beats to have a coherent structure. I share my subjective evaluation of the pieces in the following. For each piece, I used GarageBand[1] to synthesize the piano-roll. And for each piece, I chose a tempo for performance that I found suitable.

Below each piece, I try to include my subjective evaluation based on:

- Flow, dissonance, or unexpected jumps. Harmony, including key changes/modulation.
- Note value, variety in the generated note values.
- Generalization: How similar is the generated music to the training data? In other words, can I tell directly if the generated output, a musical segment, is taken from a piece in the training data or not? For instance, does it fall under a musical style similar to the style of the training data?
- Likability: Do I like the generated sound? Meaning, would I want to listen to it again?
- Structure: Does the generated output follow any form or cadence, given the short musical segments?

In Fig. 2.3 (you can listen by clicking on the images), the flow of the first generated sound is OK, meaning there is no dissonance or weird jumps, and there is a melody line. The music stays in the same key and seems to have some type of cadence at the end. It sounds like classical music to me. Overall, I think it sounds amusing, and I like it OK.

[1] https://www.apple.com/mac/garageband/

The second piece in Fig. 2.3 (you can listen by clicking on the images), also, there is no dissonance or weird jumps. There is melody line and some type of cadence. It sounds (multi-phonic) baroque style. I like it OK.

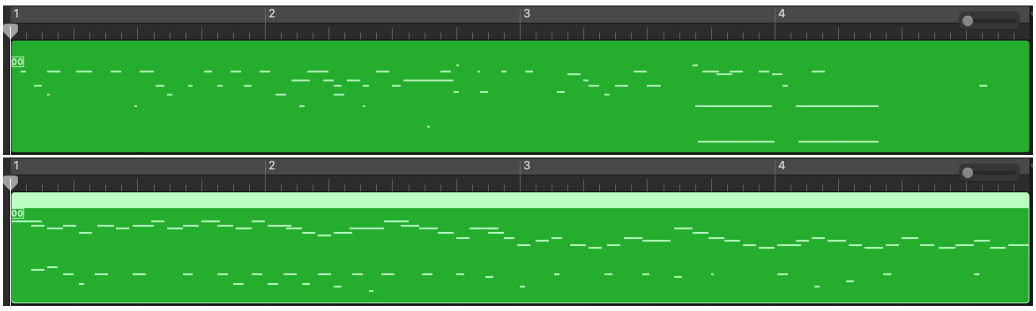


Figure 2.3. Two 16-beat long pieces generated by the model. Each piano-roll is linked to a synthesized performance by GarageBand on YouTube, you can listen by clicking on the images.

In Fig. 2.4, the presented segments are 32 beats each, even though the model is trained on only 16 beats piano-roll segments, it seems to generate some coherent longer segments. In the first audio the flow is smooth, no dissonance or weird jumps. In terms of harmony, there is a nice key change at the end and some type of cadence. Note value: there is variation in note values. It sounds similar to classical era music. I like it OK because it has some nice changes in register and a believable texture.

The second audio: It flows OK, with no dissonance or sudden jumps. There is no cadence and the music seems to modulate to a new key at the end of the segment. It sounds similar to a polyphonic baroque style piece.

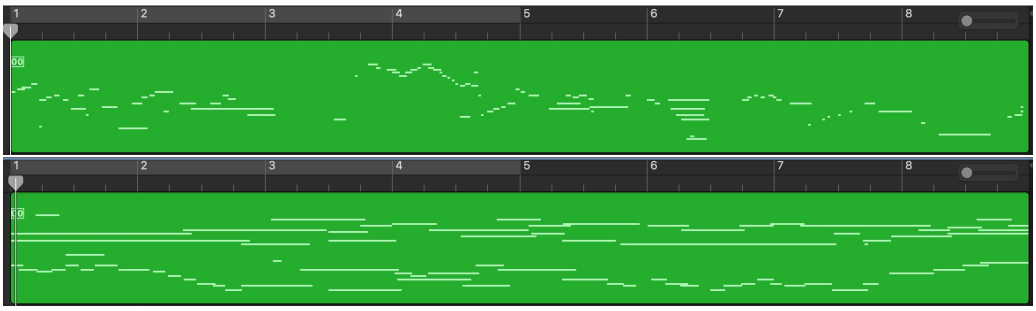


Figure 2.4. Two 32-beat long pieces generated by the model. Each piano-roll is linked to a synthesized performance by GarageBand on YouTube, you can listen by clicking on the images.

In Fig. 2.5, the generated audio segment is 160 beats long. The flow is very good and there is a melody line. The harmony sounds good with a key change. It sounds like a baroque style piece. I like the interesting structure.

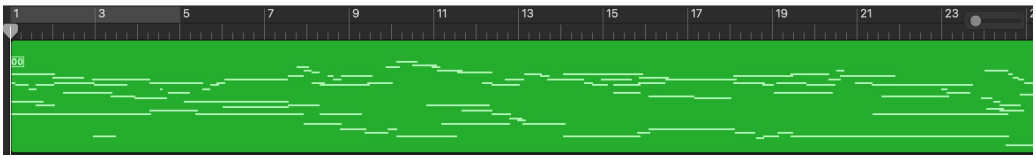


Figure 2.5. A 160-beat long piece generated by the model. The piano-roll is linked to a synthesized performance of about a minute and a half by GarageBand on YouTube, you can listen by clicking on the images.

2.10 Conclusions

The takeaways from this literature review that guide my own research in symbolic music generation using machine learning are summarized in this section. Most of the algorithmic work on generative models is focused on image and natural language, and much less on music. This could possibly be due to the commercial incentives. The novel algorithms and methods for natural language and image applications are borrowed with modification to generate music, either symbolic or waveform.

Currently the common generative models are generative adversarial networks, vector quantized variational autoencoders, transformers, and diffusion models. Transformers and diffusion models are considered simpler than the other two methods because they do not need a classifier to tell which generated entity is similar to the real entities. This simplification speeds up training and testing transformers and diffusion models compared to the other two models. While diffusion models have been shown to have multiple advantages over GANs and VAEs, using diffusion models to generate music is under-researched compared to the usage in language models and image models.

One research question is that for symbolic music, what data representations could possibly be used with diffusion models, and what are the trade-offs. In [36], the authors use the latent

space of a VAE as the data space of the diffusion model to generate symbolic music. This is possibly limiting the generative power of the diffusion model as the VAE's latent space might not be optimal for this purpose.

Another research question that is interesting to explore is how non-autoregressive diffusion models can be used for music generation applications. Filling in a partial music score and conditioning the model on rhythm are two such applications. For these applications it remains to be researched how various sampling orders and strategies differ in the quality of the produced music which is discussed in Section 2.6.2. It could also help developing a better understanding of the overall compositional stylistic structure. From my personal observation, it seems most of the used algorithms are good at generating one minute of music in a certain style but not a whole piece.

When researching and developing generative models, comparing the performance of different generative models is necessary. This turns out to be more challenging than evaluating most other ML models, for instance supervised ML models. The main downside of subjective evaluation is that measuring the diversity of the produced output is possible but impractical subjectively. Objective metrics are not that good at evaluating the artistic quality of the outputs. Therefore, both subjective and objective evaluation metrics are needed to compare methods and have an idea of the diversity and artistic quality of the generated music. The quantitative metrics evaluate the quality based on the similarity to the training data. It is possible that some model generates outputs with some being still of acceptable artistic quality but not that similar to the training data. Then the question is whether other methods for quality evaluation would generalize better regardless of the distance of the generated points to the training points.

It is apparent from the literature that the choice of the model architecture does not directly impact the quality of the generated music. However, the model architecture that is computationally more efficient and can consider more complex relations between the notes, tends to generate music that is potentially coherent up to a larger scale. Therefore, it seems there is a research opportunity to explore methods that are specific for symbolic music that can allow the

model to look at a larger range of notes.

Chapter 3

Diffusion Models

3.1 Abstract

This chapter reviews diffusion models, which are a novel approach to generative models. The introduction section presents the motivation behind diffusion models. The seminal paper that proposed diffusion models is reviewed, and some of the more recently proposed modifications of diffusion models are discussed in the following sections. And in the last part of the chapter, I discuss the use of diffusion models for symbolic music generation.

3.2 Introduction

One of the advantages of generating symbolic music compared to audio by machine learning (ML) is that manipulating the generated material is feasible. This possibility allows a composer or a musician to collaborate with ML more easily. The second reason that I have been using specifically binary piano-rolls is that the required computation for training ML models with binary piano-rolls is much lower compared to training ML models with spectrograms or even piano-rolls with dynamics. The short time it took to train a model allowed me to run several experiments in the past two months to find out what methods are hard to get to work and which ones are easier to get to work. That is the reason that in this document most of my attention is on the methods that can be applied to binary piano-rolls. Nonetheless, most of what I learned from my research so far is transferable to the other modalities.

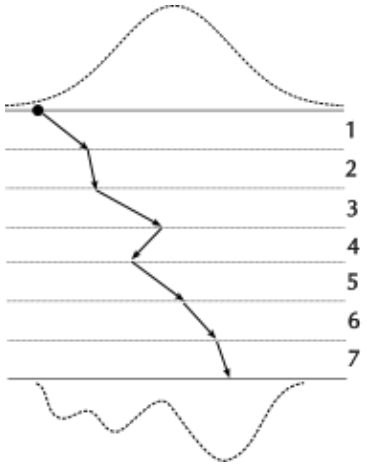


Figure 3.1. The approach used by diffusion models to learn the distribution of the data.

Machine learning generative models attempt to model the distribution of a set of data. The modeled distribution is then used to draw new samples that could pass as real data points from the true distribution.

The approaches to generative models can be divided into two broad categories. In the first category of approaches, the data distribution is estimated directly. For example, fitting a normal distribution on a set of samples falls in this category. This approach does not scale to high-dimensional and complex distributions. Meaning, to increase the accuracy of the estimation of the distribution of the real data point, that is hidden to us, an impractical amount of computation is needed. Therefore, this approach is not used to generate images, for example.

The second approach does not estimate the data distribution directly. Instead, this approach estimates a function that transforms a sample from a prior distribution to the data distribution. The prior distribution is chosen to be simple, that is its probability distribution function has no more than a few parameters. Therefore, sampling the prior distribution is fast. Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and Probabilistic Denoisining Diffusion (PDD) models or for short diffusion models fall in this category. In particular, diffusion models transform a sample from the prior distribution to the data distribution in several gradual steps. Fig. 3.1 illustrates this process by a diffusion model on an illustrative

example. A sample from the prior distribution, in this example a normal distribution, is drawn. A transformation function, which in practice is a neural network, is applied to the sample, and the transformed point at step one is returned. The point at step one is then transformed to the point at step two. This process is repeated to get the transformed point at step seven. The main claim of Probabilistic Denoising Diffusion (PDD) is that the distribution density of a large number of transformed points would be close to the true probability density function shown at the bottom of the figure. Conceptually, the prior distribution in each step morphs gradually to final distribution. For example, the distribution of the transformed points at step 4 would be half similar to the prior normal distribution and the final data distribution.

The following section reviews the paper that introduced diffusion models and explains difference between diffusion and other approaches in more detail.

3.3 Deep Unsupervised Learning using Nonequilibrium Thermodynamics

The paper that introduced diffusion probability models was published in 2015 [44]. The paper claims that the proposed diffusion model is the first method that is tractable and flexible. Models that are tractable can be analytically evaluated and easily fitted to data, e.g. a Gaussian. However, these models are unable to aptly describe structure in rich datasets. Models that are flexible can be molded to fit structure in arbitrary data. Evaluating, training, or drawing samples from such flexible models typically requires a very expensive computational process.

Learning in this framework involves estimating small perturbations to a diffusion process. Estimating small perturbations is more tractable than estimating accurately the data distribution function. Diffusion models are similar to variational autoencoders (VAEs) in that both transform the data distribution to a prior distribution (often a Gaussian), and also transform the prior distribution to the data distribution. The main differences are that the forward process (transforming the data distribution to the prior) is done arithmetically without any learning by diffusion models,

while the encoder of a VAE learns this transformation. The second difference is that the forward and backward process in diffusion models is gradual with many steps, but in VAEs the decoder and encoder in a single step transform the data points.

A diffusion model uses a neural network to transform the samples from the prior distribution as mentioned in the previous section. Therefore, some training data is required to train this neural network. The training data is collected from a process called the forward process. In the following, $q(x^{(0)})$ denotes the original data distribution. The idea of the forward process is to gradually convert the original data distribution into an analytically tractable distribution $\pi(y)$, the prior distribution, by repeated application of a diffusion transition kernel $T_\pi(y|y';\beta)$ for $\pi(y)$, where β is the diffusion rate. In the paper [44], the derivation of the kernel for the normal distribution and binomial distribution is presented. In Eq. 3.1, the shorthand for the kernel is introduced.

$$q\left(x^{(t)}|x^{(t-1)}\right) = T_\pi\left(x^{(t)}|x^{(t-1)};\beta_t\right). \tag{3.1}$$

The forward trajectory, with T steps is computed by

$$q\left(x^{(0\ldots T)}\right) = q\left(x^{(0)}\right)\prod_{t=1}^{T} q\left(x^{(t)}|x^{(t-1)}\right). \tag{3.2}$$

That is the original data distribution $q\left(x^{(0)}\right)$ is multiplied by the kernel T times to transform it to the prior distribution. The kernel $q\left(x^{(t)}|x^{(t-1)}\right)$ for a Gaussian prior is $N(x^{(t)};x^{(t-1)}\sqrt{1-\beta_t},I\beta_t)$. And the kernel for a Binomial prior is $B(x^{(t)};x^{(t-1)}(1-\beta_t)+0.5\beta_t)$. That is the kernel has the same distribution function as the prior for both of these distributions. In my experiments, to transform a piano-roll, the binomial kernel is applied to each element of the piano-roll matrix independently. For each element the output of the kernel is a binomial distribution which is sampled to get a value, zero or one. The sampled values of the elements are then placed in the piano-roll matrix. After repeating this process for a fixed larger number of steps T, the elements

of the transformed piano-roll would be samples from the prior distribution. Also, in the binomial kernel, instead of the constant value 0.5, which dictates the ratio of success or ones to failures or zeros, the average ratio of ones to zeros or number of pitches present to silence in the training piano-rolls is used.

The generative model will be trained to describe the same trajectory but in reverse. Given the prior distribution,

$$p\left(x^{(T)}\right) = \pi\left(x^{(T)}\right), \tag{3.3}$$

by repeatedly multiplying the prior distribution by the reverse kernel the data distribution is recovered,

$$p\left(x^{(0\ldots T)}\right) = p\left(x^{(T)}\right)\prod_{t=1}^{T} p\left(x^{(t-1)}|x^{(t)}\right). \tag{3.4}$$

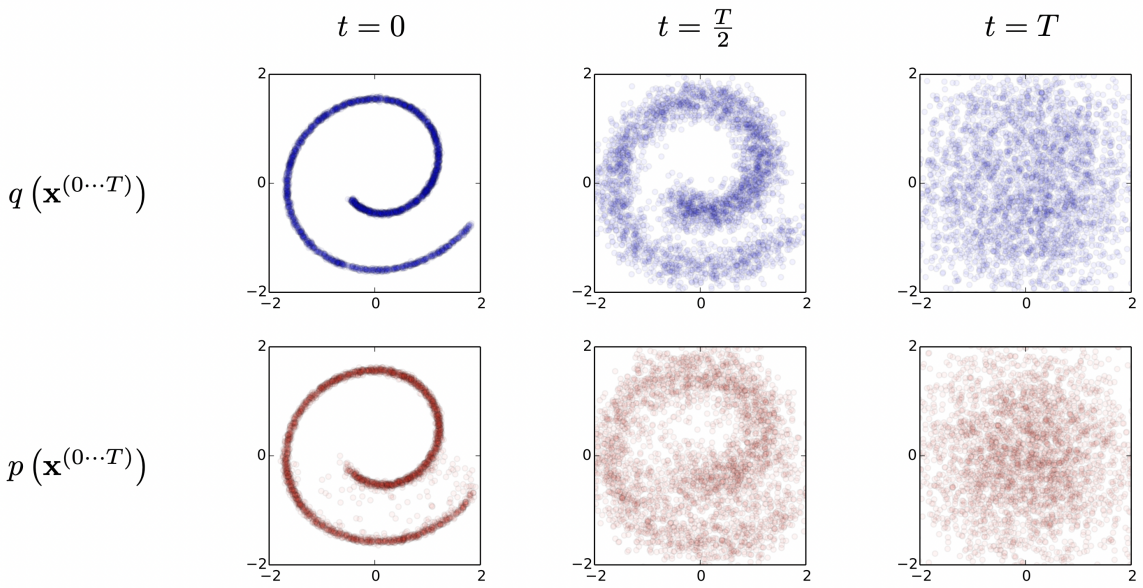


Figure 3.2. Demonstrating the forward and backward diffusion processes. On the left, the points are distributed on a 2d plane. The forward process is illustrated in the top row. The distribution of the points from the original distribution on the left is converted to a Gaussian distribution on the right in T steps. The bottom row illustrates the reverse diffusion process. The Gaussian distribution on the right is converted back to the original distribution. Source: Reproduced from [44], licensed under CC BY 4.0.

For small step size β, the reversal of the diffusion process has the identical functional form as the forward process. That means if β is small then $p\left(x^{(t-1)}|x^{(t)}\right)$ will also be a Gaussian (binomial) distribution. By increasing the number of steps T it is possible to decrease the step size β. During learning, only the mean and covariance of a Gaussian diffusion kernel, or the bit flip probability for a binomial kernel, need to be estimated. In this paper [44], multi-layer perceptrons are used to learn the kernel parameters for the reverse process. The forward process generates pairs of points, piano rolls in the case of my experiments, that are used to train the neural network. The more noisy point, from step t, is used as the input to the network and the less noisy point, from step t-1, is used as the expected output of the network. In my experiments, in each step the forward process, as mentioned before, is applied to each element of a piano-roll independently. That process results in a noisy piano-roll at each step of the forward process. Each noisy piano-roll with the less noisy piano-roll from the step before is then used to train the

neural network.

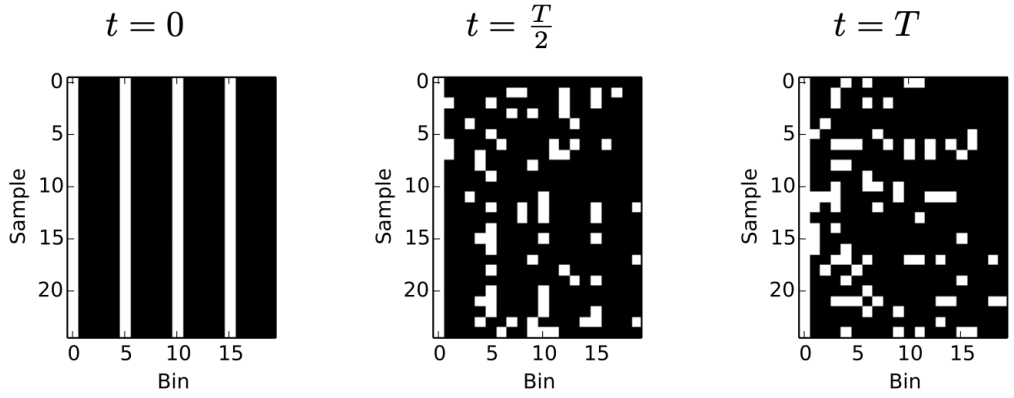


Figure 3.3. Demonstrating the conversion of a distribution to another analytical distribution. On the left, 25 samples of a binary pattern of length 20 are presented. That is, there are 25 data points, and each data point is a vector with 20 elements. In the picture, all 25 vectors are shown in the rows. The black bins represent the value of one, and the white bins represent the value of zero. Using an independent binomial noise distribution as the analytical distribution, on the right. On the left, the generated samples are identical to the training data. The white column on the left of the three pictures is not part of the data and is added by the authors to the plots to have enough white space to see the ticks on the vertical axis. Source: Reproduced from [44], licensed under CC BY 4.0.

Fig. 3.2 illustrates the forward and reverse diffusion process with 2-d swiss roll data transformed to a Gaussian distribution. New samples from the Gaussian distribution are then transformed back into the data distribution using the reverse process. Fig. 3.3 presents another example with a Binomial distribution. The input data is binary and the Binomial kernel repeatedly flips the bits randomly until at the step time T, the data distribution is transformed to the Binomial distribution.

3.4 Modern Diffusion Models

The authors in [45] proposed a new equation for the Gaussian kernel in the forward process that depends only on the input x_0. This has two benefits. First, to compute the output at time step t there is no need to compute the output at the time steps before t. This way the output at random ts can be computed and used to train the neural network for the reverse process. That

means with the kernel from the original paper [44], the whole set of noisy training data should be generated and stored in memory. This is needed to shuffle the order of noisy training data in mini-batches which is required for training neural networks using mini-batch gradient descent. With this updated kernel, it is possible to first pick a random order for the noisy piano-rolls and then generate them on the fly without generating the preceding noisy piano-rolls. Therefore, prior to training the network, it is not necessary to generate all the noisy piano rolls, as was done in the original 2015 diffusion models paper. As a result the amount of memory required is just the needed amount to hold a single mini-batch and not the whole training set for the neural network. Second, this simplification of the kernel equation simplifies the loss function for training the neural network. The derivation of the simplified Gaussian kernel is presented in the following by

.

$$\alpha_t = 1 - \beta_t \tag{3.5}$$

$$\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s \tag{3.6}$$

$$q(x_t|x_0) = N(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) \tag{3.7}$$

Using the same definition of α_t and $\bar{\alpha}_t$ it is also easy to write the binomial kernel to depend only on the input x_0,

$$q(x_t|x_0) = B(x_t; \bar{\alpha}_t x_0 + (1-\bar{\alpha}_t)0.5). \tag{3.8}$$

Therefore, it is possible to just change the diffusion rate β_t schedule and have a binomial kernel to depend on the input x_0. As discussed later in this section, another modification in the diffusion process decouples the forward process from the reverse process. As a result, the schedule for β_t does not have to be a certain function.

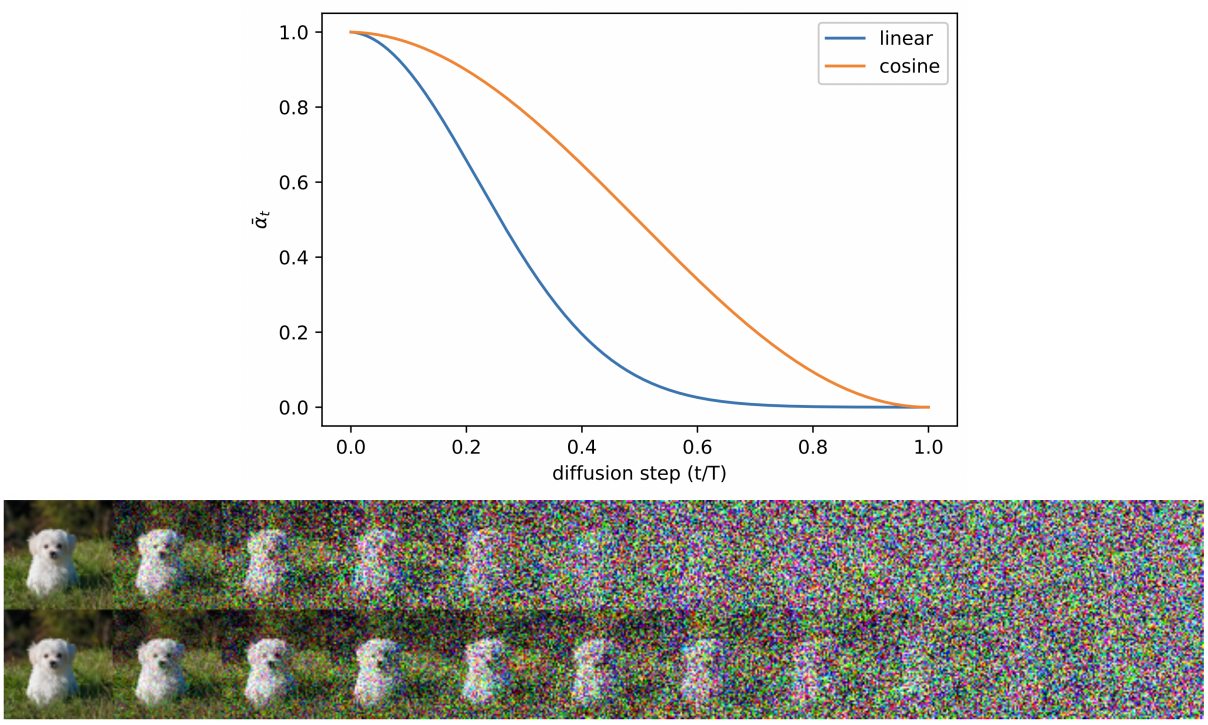


Figure 3.4. The difference between linear and cosine forward diffusion weighting. Source: Reproduced from [1], licensed under CC BY 4.0.

Another improvement proposed in [1] is changing the diffusion rate schedule. The authors propose instead of a linear schedule, a cosine schedule to be used for the diffusion rate. The two schedules linear and cosine are shown in Fig. 3.4. It is seen the cosine schedule reduces α more slowly. The motivation for this improvement is presented in Fig. 3.4-bottom. In the top row, with the linear schedule, the image is pure noise at the last four steps. Which means some computation time is wasted, and the neural network has a harder time to learn from those steps. In the bottom row, with the cosine schedule, some information from the image is still visible until the last one or two steps.

Experimenting with different diffusion rate schedules remains an interesting aspect to explore in my experiment to see whether it is possible to reduce the number of training samples and as a result train the neural network in a shorter time. The rate schedule that I ended up using that is simple and yielded good results is $\beta_t = t/T$.

Another major improvement was presented in a paper in 2020 [46]. The proposed

modification to the sampling process is to predict x_0, denoted by $\hat{x}_0$ directly from x_t. Then add the noise corresponding to step $t-1$ to $\hat{x}_0$ to obtain x_{t-1}. And repeat this process up to T times. In the original diffusion model the reverse process has to have the same number of steps as in the forward process as each step in the reverse process remove the same amount of noise added in the corresponding forward step. The method proposed in this paper in effect decouples the reverse process from the forward process. The paper shows by reducing the number of steps in the reverse process to an arbitrary number less than T, it is possible to trade a small amount of degradation in accuracy or quality of generated images to gain generation speeds of up to 100 times. Fig. 3.5 compares this sampling process to the original sampling process by diffusion models. In my own experiment, I ended up using this process as it turned out to be more stable, meaning the generated piano-rolls had a higher quality when I adopted this approach compared to the original diffusion approach with only 100 diffusion steps.

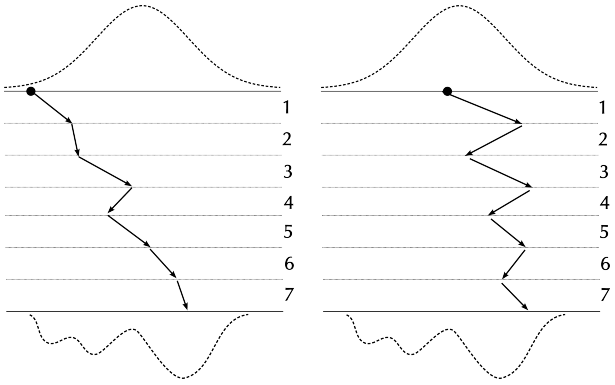


Figure 3.5. Left, the original sampling process denoises a sample drawn from the prior distribution in each step. Right, the sampling process that alternates between estimating the noiseless point from the data distribution, and adding some noise less than the previous step. The odd numbered rows correspond to the steps that predict x_0, and the even numbered rows correspond to the steps that add back some noise.

3.5 Conditional sampling and latent space

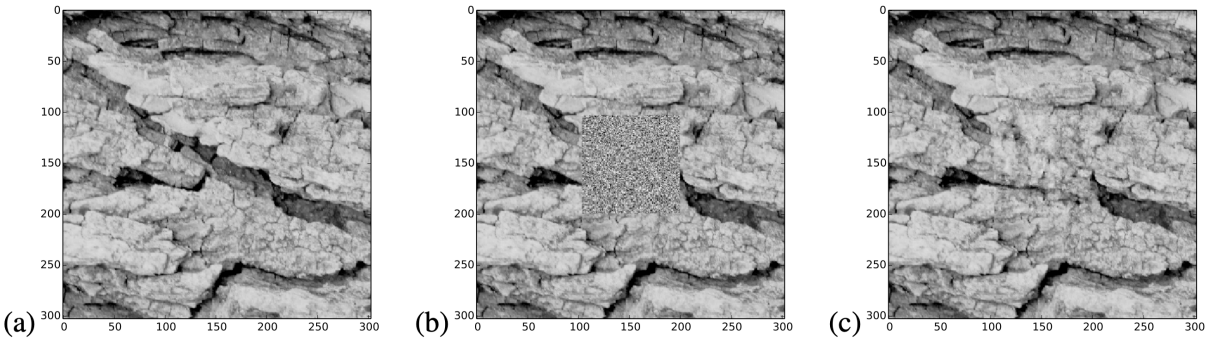


Figure 3.6. Demonstrating image inpainting using a diffusion model. (a) a bark image from a dataset of images. (b) the same image with the central 100×100 pixel region replaced with isotropic Gaussian noise. This is the initialization $\tilde{p}\left(x^{(T)}\right)$ for the reverse trajectory. (c) the central 100×100 region has been inpainted using a diffusion probabilistic model trained on images of bark. Note the long-range spatial structure, for instance, in the crack entering on the left side of the inpainted region. Source: Reproduced from [44], licensed under CC BY 4.0.

In these early diffusion papers, multiple applications are presented that inspired some of my experiments with piano-rolls and diffusion models. Inpainting is presented in Fig. 3.6. This is similar to the application Coconet [10] was designed for. That is, a score (piano-roll) is partially provided by the user as a prompt and the model is supposed to fill the rest of the score. This application is possible as diffusion models can be conditioned on any part of the input. This is done by excluding the prompt part of the input from the process that adds noise. The network prediction for the prompt part is also ignored in each of the sampling iterations.

Another application that was explored in [45] is generating similar points in the data space. A face image is taken to the latent space of the diffusion model, then the reverse process can generate multiple images that are similar. This fact is used in my experiments to generation variations of a piano-roll segment. By increasing or decreasing the noise level of the initial point in the sampling process, it is possible control the degree to which the generated piano-rolls differ from each other.

Another interesting experiment is explored in [45], where two face images are converted

to their latent representation using a diffusion model. Then in the latent space new points are linearly interpolated, which are converted to face images using the reverse diffusion process. The generated face images have shown a small inconsistency, meaning they have less obvious artifacts due to the interpolation in the latent space versus the interpolation in the input space.

3.6 Neural network architecture

The UNet architecture was introduced in 2015 for image segmentation [47]. It resembles the typical architecture of autoencoder, called often the hourglass architecture, that in multiple steps subsamples the feature-maps in the first half of the network and then in the second half the featuremaps are upsampled to finally have the identical size of the input. The difference between UNet and hourglass is that UNet has horizontal connections that connect the feature-maps at the same scale from the first half of the network to the second half. The UNet architecture that is used in my experiments is shown in Fig. 3.7.

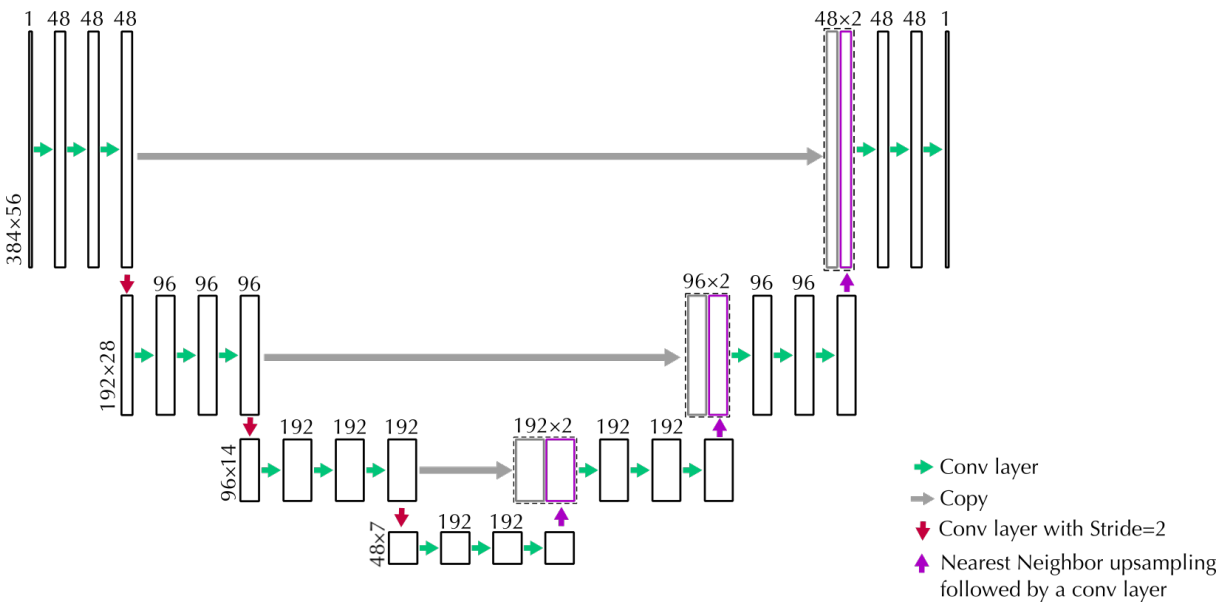


Figure 3.7. The UNet architecture used in my experiments. The input piano-roll is a matrix of size 56×384. Each convolution (conv) layer consists of several convolution filters. Each green arrow shows a conv layer. The rectangles depict the featuremaps, with the number above each rectangle showing the depth of the featuremap tensor, which is also the number of conv filter in the corresponding layer. The red arrows show the conv layer with a stride of two that reduces the size (rows and columns) of the featuremaps by 2. The purple arrows show the upsampling step using the nearest neighbor interpolation, which doubles the size of the featuremaps. The grey arrows depict copying a featuremap from the left side of the network to the right side. On the right side, the copied featuremaps are stacked on the upsampled featuremaps, the result is a featuremap with double the depth of the featuremap tensor below. The last conv layer at the top right contains a single convolution filter; as a result the output is a single piano-roll. The number of rows and columns of the tensor in each row or level of UNet remains the same due to padding the input when applying convolution filters.

Due to the horizontal connection, if the UNnet network is asked to learn reconstructing the input as in a regular autoencoder then UNet would simply learn the identity function using only the first row of layer and without using the higher scale layers at the lower layers. Therefore UNet cannot be used in such a case in place of an hourglass network. But in a noise reduction setting where the network is given an input with added noise and is expected to remove the noise, a UNet network can potentially use features at all the scales in the network to denoise the input. An hourglass network in this case would use only the featuremaps at the highest scale to denoise the input.

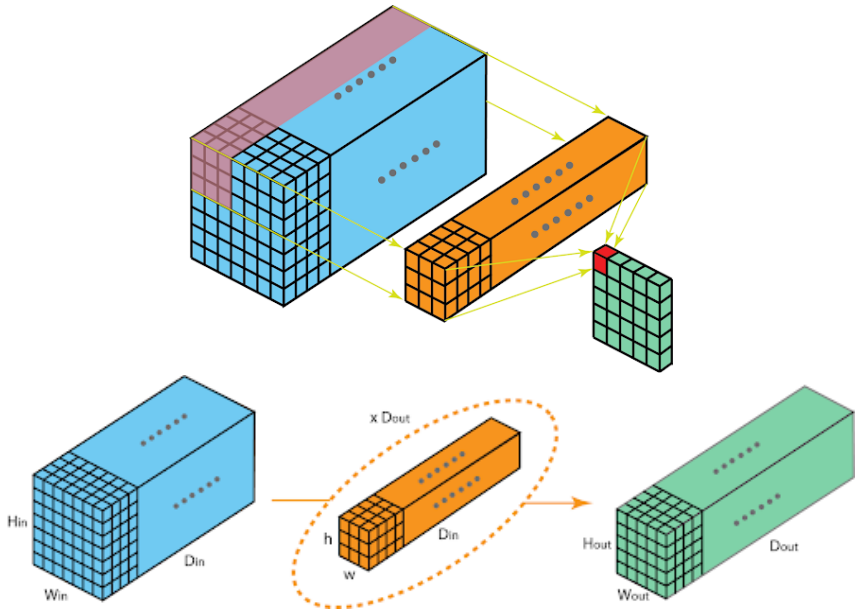


Figure 3.8. The top figure shows a tensor in blue and a convolution filter in orange, applying the conv filter, that is overlapping the filter tensor on the input tensor and multiplying them element-wise, and then shifting the filter and repeating, results in a single matrix or a 2d tensor shown in green. In the bottom figure, Dout filters are applied to the input tensor. The result is Dout matrices, or a tensor with the depth of Dout. Therefore, in a conv net to have feature-map with a certain depth size that number of conv filters are required. In the figure of the UNet architecture used in my experiment, the numbers above the rectangles of the feature maps, hence, tell the number of filters in each conv layer shown by a green arrow before the feature-map.

There are three main sets of operations in UNet. First, the convolution (conv) layers containing convolution filters take as input a tensor - a term used to refer to the generalization of a matrix with height, width and depth in the ML literature - and produce an output tensor. Each filter itself is a tensor of the depth of the input tensor but with a small number of rows and columns, typically 3×3 to 5×5. The filter tensor is overlaid on the input tensor, the two tensors are then element-wise multiplied and the result is summed to a single scalar value. Then the filter tensor is shifted horizontally and also vertically to have its center overlap with every element in the rows and columns of the input tensor. By padding the input, the output matrix of this operation can have the same number rows and columns as the input tensor. The output of a conv filter is also called a feature-map as each filter detects patterns also know as features and the output indicates how strong is the presence of the feature at each element. The convolution

operation is illustrated in Fig. 3.8. As each conv filter would produce a tensor with the depth of 1, n conv filters are needed to produce a tensor with the depth of n.

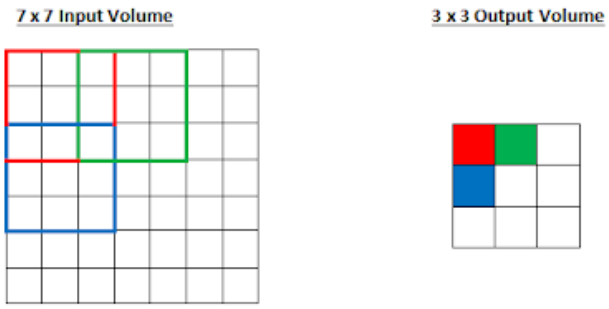


Figure 3.9. The figure depicts down-sampling a feature-map by using conv filter with stride of 2. On the left, the same filter is shown that is shifted two element to the right and then also shifted two elements to the bottom, from the top left corner. At each position, the usual element-wise multiplication of the elements of the input matrix and the filter is computed and summed to a single scalar. The output therefore has half the number of rows and column of the input matrix. As the network during training adjust the weights of the filter to reduce loss, the learned filter weights are expect to preserve relevant information in the output; as a result, this method of using conv filter with stride of 2, compared to a simple decimation-based downsampling, preserve more relevant information while reducing the size of the feature-map.

The second main operation in UNet is downsampling the featuremaps. Downsampling is done by using conv filters with stride of 2, skipping every other element of the input tensor horizontally and vertically. Fig. 3.9 illustrates this operation. Using a conv layer with stride of 2 to downsample feature-maps has an advantage over just dropping or decimating every other column and row. The learnable weights of the conv filters have an opportunity to generate featuremaps that preserve information while reducing the size of featuremaps.

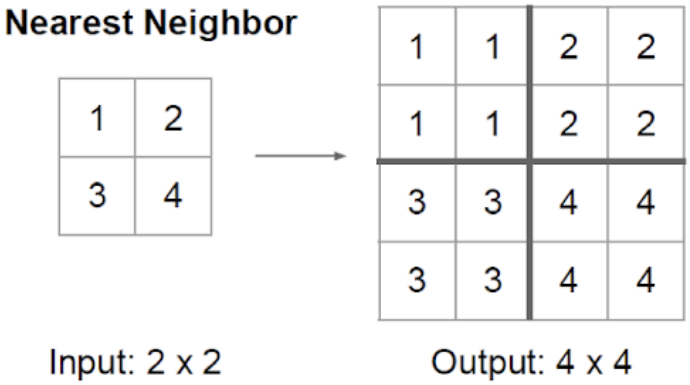


Figure 3.10. The upsampling operation used in UNet is based on simple nearest neighbor interpolation. This is simply duplicating each element in the input feature-map 3 times to double the number of rows and columns of the output feature-map.

The third main operation in UNet is upsampling the featuremaps. On the right side of the network, the featuremaps are upsampled from higher scales to be combined with the featuremaps at lower scales. The upsampling process simply duplicates the values in the input tensor. This process is illustrated in Fig. 3.10. Upsampling brings the information from higher scales to lower scales, which is used then to guide the finer details.

There are several aspects of the network architecture that I intend to explore and optimize for the piano-rolls. For instance, what is the minimum number of convolution layers and convolution filters in each layer in the UNet that would produce the most coherent piano-roll it can generate at all scales? Whether the left side and right side of the UNet need to be symmetric in the number of layers and filters? and whether an asymmetric UNet would be able to use the computational resources more optimally? And more music theory related questions such as whether it is possible to have asymmetric accuracy and be more accurate for pitch and less accurate for time length of the notes? This is for the obvious reason to not waste computational resources on being very accurate to one or two ticks, for instance. However, it is important to ensure that it is not off by even a single MIDI pitch. Thinking that possibly harmony, notes and relationships between notes is more valuable or time or meter, musically speaking. It could be this asymmetric accuracy could be introduced through the loss function. Another similar

question is whether it is possible to change the piano-roll representation or the loss function to assist the network to some degree with learning the distance between pitches and intervals in a more musically meaningful way? These are sort of questions I will be working on along with the main ideas discussed in the last section of this document.

3.7 What the network perceives

One question that comes to mind is whether the UNet network loses too much information along the pitch axis of the piano-rolls each time the input is downsampled. Given UNet architecture is designed for processing images and not piano-roll, it is a possibility some aspects of the architecture are just bad for processing a piano-roll with time along one axis and pitch along another versus an image that has identical units for the horizontal and vertical axes.

A method that, to some degree, reveals what each convolution filter in a network is looking at is Activation Maximization (AM) [48]. The algorithm of AM is presented below:

```
1. take a random input. In my experiments,
    this is a piano-roll sampled from a binomial
    distribution.
2. Pass the input to the network.
3. Calculate gradient of a particular convolution
   filter's average output with respect to input.
4. Update the input with gradient ascent. This
   in result increases the average of output
   matrix of the convolution filter.
5. Go to step 2, repeat for N times.
```

In my experiment I found after about N=200 iterations the updated input does not change much. Therefore used N=200. The output of AM gives a general idea of what each of those

filters is looking for in the input image. For example, in the left image multiple frogs partially indicating the corresponding filter has learned to detect frogs in images.

Applying activation maximization to three convolution filters of the last convolution layer at the bottom of UNet that is trained in my experiments, Fig. 3.11, reveals the convolution filters have a high resolution view of the input piano-roll. Therefore, the three downsampling steps on the left side of UNet are not losing information along either pitch or time axis. If that was the case, and the convolution filters had a low-resolution view of the piano-roll, the output of AM would have multiple elements along the time or pitch axis with the same color in the repeated patterns. This confirms using convolution filters with stride of two to downsample the feature-maps does not lose information as one expects from a simple decimation down-sampling method.

Interpreting the AM generated piano-rolls does not seem as easy as the images generated for computer vision algorithms. It remains an interesting question to apply AM to the later convolution layers in the piano-roll UNet and whether there is a modification that can make the AM outputs more interpretable for binary piano-roll.

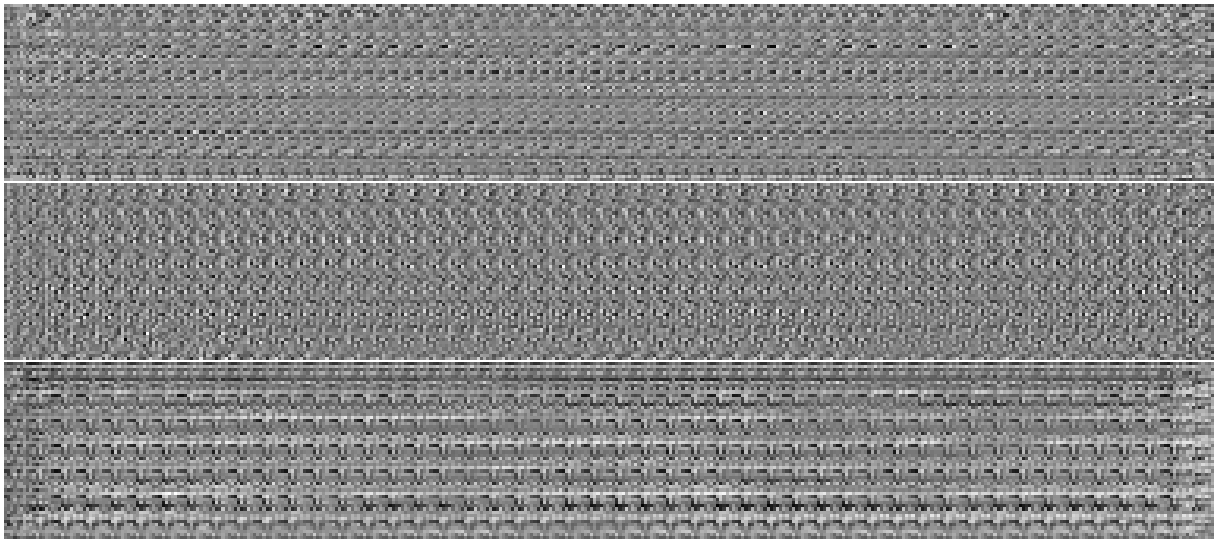


Figure 3.11. Using AM, the input that maximizes the activation of three convolution filters at the lowest resolution of the UNet trained on piano-rolls are visualized. The visualized paino-rolls reveal the filters at this level, after three downsampling steps have a high resolution of the input piano-roll without losing information along the pitch or time axes.

3.8 Experiments

To train the diffusion model, I used the Maestro dataset that contains MIDIs and audio files of recorded performances from piano performance competitions. In the dataset there are just over 2,000 MIDI files and the title and composer name for each MIDI is in a text file. I went through the list and removed the performances of the same piece by keeping only one of the MIDI files per title. At this point, about 500 MIDI files are left out of 2,000. Then using a Python script, I found that there are 78 MIDIs that have pitches only between 33(A1) and 88 (E6) MIDI notes (56 MIDI notes).

The 78 MIDIs were converted to piano-rolls. The resolution used in the conversion was 1 beat in the MIDI file converted to 24 ticks. Each piano-roll is then divided into non-overlapping segments of 16 beats or 384 (16×24) ticks. This process at the end yielded 2,044 piano-roll segments that were used to train the model.

The number of diffusion steps, T, used in my experiments was set to 100. The UNet neural network was trained on 50 epochs. In each epoch, each one of the 2,044 piano-roll segments is used to generate 100 noisy piano-rolls that have vary from no noise to all the way being a sample from the binomial distribution. Therefore, in each batch there were 204,400 training samples. Training the UNet on the 50 epochs took about 48 hours on two NVIDIA A600 GPUs.

One of the interesting aspects of diffusion models is that they can be used as generative models conditioned on any part of the piano-roll. For instance, it is possible to prompt the model with half the piano-roll and the model generates the remaining half. To achieve this, the sampling algorithm is modified by excluding the part containing the prompt from the process that adds noise to the piano-roll. A sample piano roll is initially drawn from the binomial distribution. Replace the beginning of the noise sample with the prompt. The network takes this piano-roll and attempts to remove the noise. Then a smaller amount of noise is added to piano-roll. After that the original prompt segment is placed back on the noisy piano-roll and the process repeats.

At the end, the model generates a piano-roll that is coherent with the prompt. The following two examples demonstrate this: Fig. 3.12 provides both listening (click to listen) and piano-roll examples.

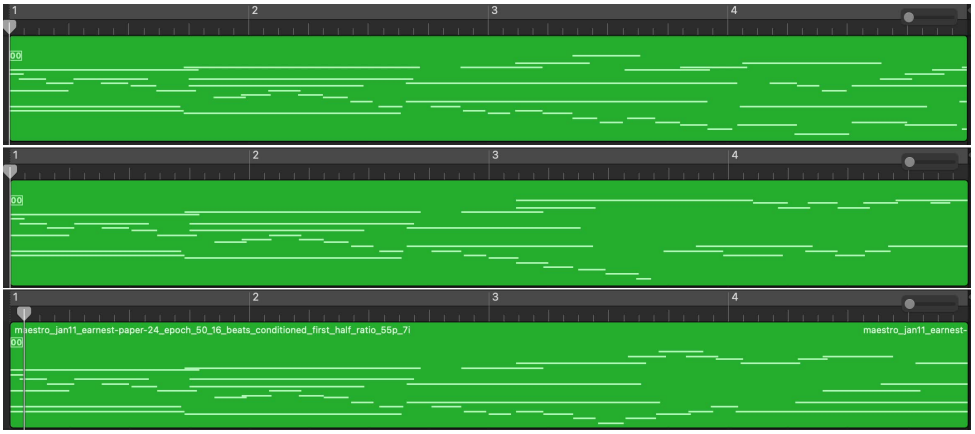


Figure 3.12. Top: original piano-roll segment. Middle and bottom: the diffusion model is prompted with the first half of the original piano-roll and the second half is generated by the model. As the sampling process is stochastic, many samples can be generated using the same prompt. Here, two samples are presented. Each piano-roll in this figure is linked to the synthesized audio.

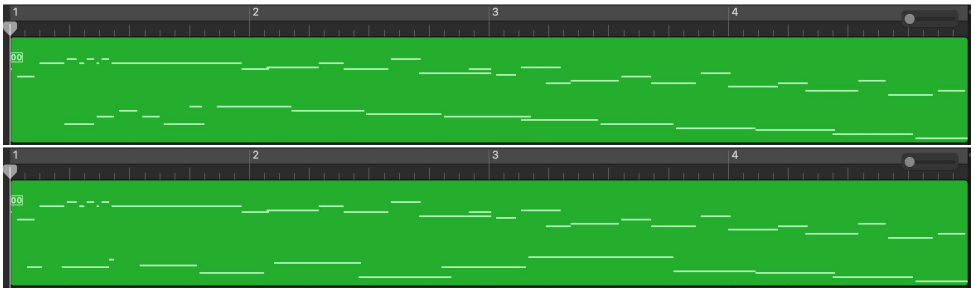


Figure 3.13. Top: original piano-roll segment. Bottom: the model is prompted with the high register pitch and the low register pitch is generated by the model. Each piano-roll in this figure is linked to the synthesized audio.

It is also possible to prompt the network along the pitch axis. For example, one can provide a melody line to be harmonized by the model. Here is an example of this experiment where the high register pitches in the piano-roll are kept as the prompt and the lower register is being generated, Fig. 3.13.

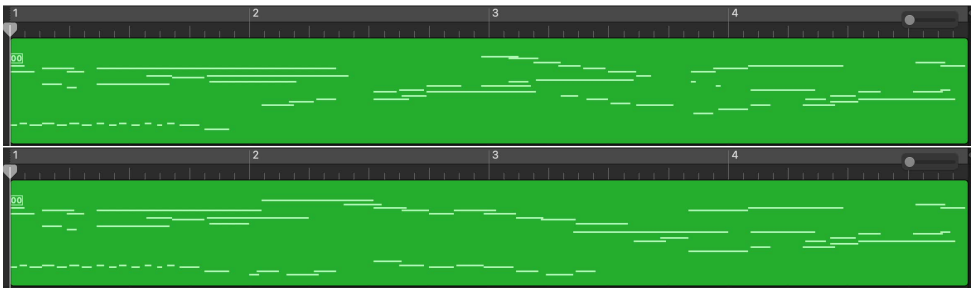


Figure 3.14. Top: original piano-roll segment with the first and last quarter used as prompts to the diffusion model. Bottom: the model fills the middle half of the piano-roll to be coherent with the first and last quarter. Each piano-roll in this figure is linked to the synthesized audio.

An experiment similar to the previous one uses the conditional aspect of diffusion models to prompt the beginning and end of a piano roll, allowing the model to fill in the middle segment. I thought this would be an interesting experiment to see how close would the generated style be to the original score composed by the composer. The process is similar to the previous experiment, where the prompt segments are overwritten on the piano roll at the end of the sampling iterations. Fig. 3.14 presents a generated piano-roll using this process.

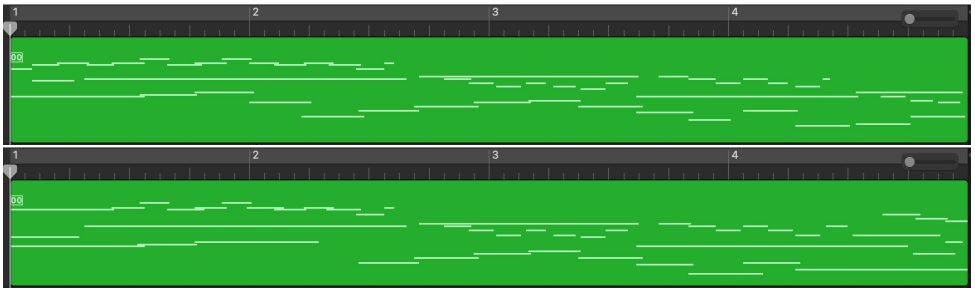


Figure 3.15. Top: original piano-roll segment. Bottom: a variation of the original piano-roll generated by the diffusion model. Each piano-roll in this figure is linked to the synthesized audio.

Another aspect of diffusion models is that in the latent space, which is the noisy piano-roll in my experiments, the pieces that are similar sounding are close in the latent space. Therefore, it is possible to add noise to a piano-roll, for instance, the noise level of step 80 out of 100 in the forward process, and then denoise the piano-roll using the diffusion model sampling method. The difference is that the starting point of the sampling algorithm is not a sample from the binomial

distribution, but it is a piano-roll with added noise at the level corresponding to the step 80 of the forward process, for instance. The output would have similarities to the original piano-roll and can possibly be called a variation of the original piano-roll. As the sampling process is stochastic, it is possible to generate multiple variations of a single piano-roll. An original piano-roll and a generated variation of it are shown in Fig. 3.15.

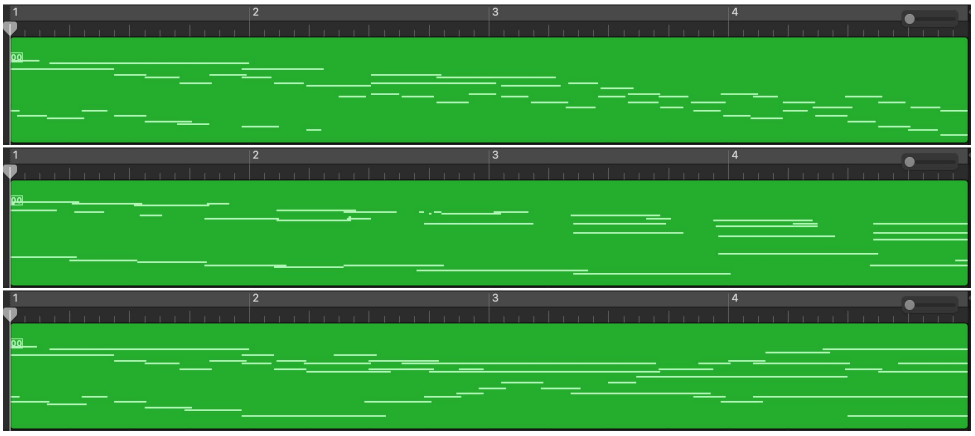


Figure 3.16. Top and middle: two original piano-roll segments that are used to generate a new piano-roll that sounds similar to both. Bottom: the diffusion model is used to interpolate a piano-roll between the two original piano-rolls. The interpolation process is explained in the text. Each piano-roll in this figure is linked to the synthesized audio.

It is possible, as mentioned in the previous section, to interpolate between points in the latent space of diffusion models with a Gaussian prior. With a binomial prior it is less straightforward to decide what distance metric to be used for interpolation. The method I tried with the piano-rolls is to add binomial noise corresponding to the step 70 out of 100 to two piano-rolls. Then randomly half of the elements from one noisy piano-roll and the other half from the other noisy piano-roll are used to create a new piano-roll. Finally, using the diffusion sampling method the interpolated noisy piano-roll is denoised to generate a new piano-roll that is expected to have similarities to the two original piano-rolls. An example of an interpolated piano-roll is presented in Fig. 3.16.

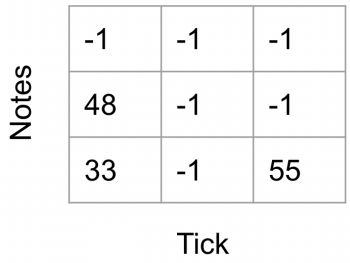


Figure 3.17. Reducing the size of a piano-roll is possible by reducing the number of rows, and indicating what MIDI note each element of the piano-roll play. -1 indicates the note does not play any note. The main change required with this representation is using a multinomial distribution instead of a binomial distribution with the diffusion model.

3.9 Research direction

To reduce the computational demand of the neural network training process, it is possible to use a more compact piano-roll representation. By setting a limit on the maximum number of pitches present at each tick, 10 pitches for example, each element in the piano-roll then would have a value indicating the MIDI note present at that time tick. And the value of -1 would indicate the element does not play any note. This compact representation, I think, may allow the diffusion model to work with much longer piano-rolls. The simplified example in Fig. 3.17 illustrates the idea.

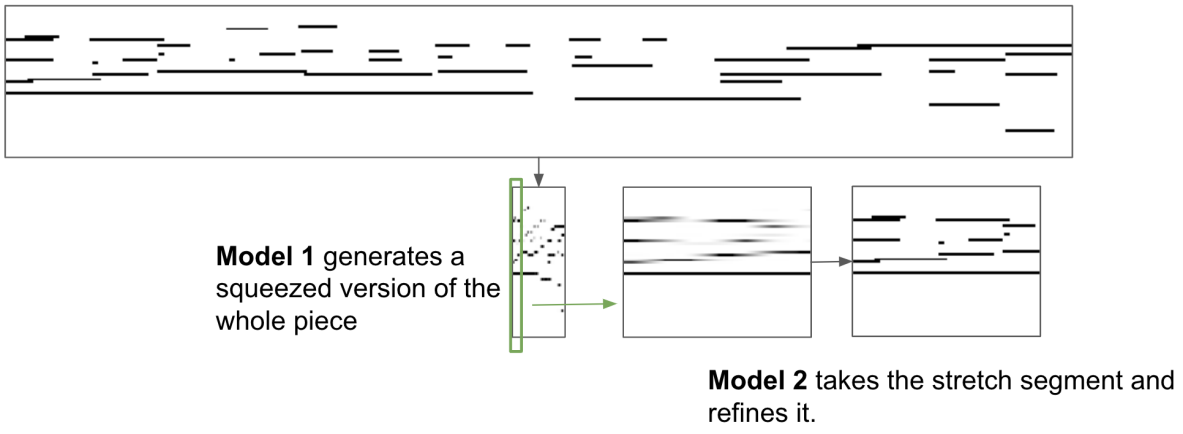


Figure 3.18. Depicting the two step process to generate a complete piece with coherent form.

One of the main research directions that I would like to explore is how to develop the

method I have worked on to generate complete piece that have a similar form to the pieces in the training set. An idea that I am going to try is having a two step process generating a piece. This process is illustrated in Fig. 3.18 . In the first step, a compressed in time version of the piece is generated. This is done by training a diffusion model on the pieces in the training set, which are compressed in time by about 40 times. This ratio 40 to 1 would make the pieces short enough that are the same length as the current piano-roll segments I am using in my experiments. At sampling time, the output of first diffusion model is a compressed piano-roll which is then stretched back to the original size, 40 times longer, by linear interpolation.

In the second step, a segment of the generated piano-roll from the first step is given to a second diffusion model. The second model refines the input piano-roll segment adding the details that were removed during the process of compressing and stretching. All the generated and refined segments are stitched together preserving order to create the complete piece. After experimenting with this method, it is possible to investigate how much of the coherence at large scale and music form are present in the generated pieces.

Another idea that I am going to work on is adding dynamics to the generated piano-rolls. The MIDI files in the training set I am using already have dynamics as MIDI note velocity. Probably modifying the diffusion model to use multinomial noise instead of binomial noise is the only major change needed to add dynamics to the generated piano-rolls. By quantizing the MIDI velocity to a smaller range, for example from 127 to 10, it is possible to reduce the extra computation needed due to having a multinomial distribution.

While working on symbolic music generation is less computational resource intensive and has some unique applications, generating music in the audio form directly makes it possible to generate music that cannot be transcribed in a piano-roll or score. Some of the ideas I am working on are probably transferable to audio music generation as well. Therefore, by exploring music generation in audio form, it is possible to try these ideas in that domain and come up with new ideas as well.

Given there are several options for the network architecture and operators in the network,

and each has its own inductive biases, experimenting with different network architectures and operators is also something I am planning to work on. Ideally, I would be able to pick choices that make more sense for piano-rolls, and may also be able to modify some to be more efficient or compatible with piano-rolls.

3.10 Conclusions

In my experience, diffusion models are simple to train and can learn and generate patterns in the data that sort of make sense. ML based generative models seem to become more impressive with more training data. That is why I am thinking to work on training a diffusion model on a training set that is 10 times larger and compare the music my current model generates with the model trained on the much larger data. That experiment will probably help me have a better idea of the limits of this approach in understanding musical patterns in piano-rolls.

Several aspects of the diffusion method can be further optimized for music generation, particularly for symbolic music, as I discussed in this chapter. One of my primary goals is to use generative models to create entire long-form pieces of music that remain coherent across all scales, rather than just short segments.

Chapter 3, in part, is a reprint of the material as it appears in Sound and Music Computing Conference Proceedings 2023, Atassi, Lilac. The dissertation/thesis author was the primary investigator and author of this paper.

Chapter 4

Large Language Models: From Notes to Musical Form

4.1 Abstract

Recent music generation methods based on transformers have a context window of up to a minute. The music generated by these methods is largely unstructured beyond the context window. With a longer context window, learning large-scale structures from musical data is a prohibitively challenging problem. This chapter proposes integrating a text-to-music model with a large language model to generate music with form. I discuss some solutions to the challenges of such integration. The experimental results show that the proposed method can generate 2.5-minute-long music that is highly structured, strongly organized, and cohesive.

4.2 Introduction

With the most recent generation of generative machine learning (ML) models, a revived interest in ML-based music generation methods has emerged. Generative Adversarial Networks (GANs) [49] and Variational Autoencoders (VAEs) [50] were among the earliest methods in this recent wave of new generative models. The computational cost of GANs and VAEs is prohibitive for learning long sequences like text. This is mostly due to the computational cost of convolution filters. The transformer architecture [51] led to multiple approaches to

generative models for text. The attention layer in the transformer architecture proved to be more efficient than convolution filters for learning relationships in long sequences. Although, with some modifications, it has been shown that convolutional networks can be as efficient as transformers [52, 53], transformer-based network architectures have remained the most common in the literature.

The transformer architecture, for instance, is used by T5 [54] to train a single model on multiple text-to-text transformations, including translation and question answering. The original transformer model is adapted by two prominent architecture approaches. The first approach is an encoder-only architecture, which is used by Bert [55] and a family of similar models including Roberta [56] and Albert [57] to encode text for classification tasks. Such models are trained to estimate $P(t_k|t_0,\ldots,t_{k-1},t_{k+1},\ldots,t_n)$, probability of the input at index k given the rest of the input.

The second approach is a decoder-only architecture which is used by GPT [58] to generate text. The model is trained to estimate $P(t_k|t_0,\ldots,t_{k-1})$, the probability distribution of the input at the last index given the prior indices. This autoregressive model is simpler than the model used by Bert, requiring fewer parameters. Consequently, the less computationally complex model of GPT can be designed to process longer input sequences. In the GPT family of models, GPT-1 to GPT-3, the input sequence length is increased and the models are deeper, which, combined with larger datasets, leads to improved performance. An initial test by [59] on people's ability to tell whether a 500-word article was written by humans or GPT-3 show a mean accuracy of 52%, just slightly better than random guessing.

All the new wave of generative models have been explored in the literature to generate music. The generative models have been applied to both symbolic [60, 61, 62] and audio music [63, 64]. Most of the recent larger models are trained on audio [65, 66], as collecting a large training set of music audio is more readily than symbolic music. Another argument is that generating audio directly can be more expressive than using a synthesizer to convert generated symbolic music into audio [67].

Music Transformer [68] adapted the decoder-only transformer architecture for music generation, capable of processing the number of input tokens equivalent to up to one minute of music. The MIDI data is transformed into a sequence of events to be more compact and suitable for the transformer. To reduce the required training data, in [34], the input attention layer is modified to have relative positional encoding, which is based on using several regular positional encoders. The larger number of positional encoders increases the computational cost of inference and training. The generated music sometimes would show structures that are musically coherent up to the scale of about one minute.

Jukebox [69] uses transformers in the latent space of a multi-scale variational auto-encoder (VAE) to generate raw audio music. Three separate VAEs are trained on the audio data, with context windows of 24, 6, and 1.5 seconds, to compress the audio at three levels. The latent vectors of the VAEs are quantized, with a codebook size of 2048 for each level. Three separate autoregressive transformers are trained. The top level transformer predicts the tokens for the top-level, with the prior top-level tokens as the input. The middle-level transformer predicts the middle-level tokens, with the prior middle-level and top-level tokens as the input. The bottom-level transformer predicts the bottom-level tokens, with the prior bottom-level and middle-level tokens as the input. To generate music, the predicted bottom-level tokens are decoded using the decoder of the bottom-level VAE.

More recent music generative models [65, 66] are based on the same building blocks as Jukebox. The main development is the simplified architecture. The recent models, including MusicGen that is discussed in more details later in the chapter, use a single scale, non-hierarchal, model to compress the audio. And the latent vectors of the encoder are quantized using Residual Vector Quantization (RVQ) which is more efficient than multiple VQs. A single transformer is trained in the encoder's latent space. For example, MusicGen, with its simplified architecture, can generate one minute of music in about two minutes, while JukeBox takes around 10 hours to generate the same duration. Of course, the generation time depends on the hardware, but the near two orders of magnitude in speed improvement is due to the simplified and more efficient

architecture.

Musical form is a multifaceted and complex concept. For the purpose of this chapter, a simple definition can help clarify the limitations of the proposed method. Form can be seen as the framework that unifies a piece of music, giving it a sense of cohesion. Some forms define rigid musical structures with distinct parts or segments, while others are without any clear part boundaries. The mentioned methods, including JukeBox, Music Transformer, and the diffusion-based method of Chapter 3, can generate music pieces longer than the training music segments in a sliding window fashion. But this always leads to either meandering and directionless or highly repetitive music. Therefore, a music generative model that can generate music in various forms is more desirable and can potentially generate music similar to musician-composed pieces.

In the rest of the chapter, after reviewing a text-to-music and large language model, I present my approach to integrating a text-to-music model with a large language model to generate coherent and structured music using Large Language Models (LLMs). The experiments and evaluations are presented before concluding the chapter in the final section.

4.3 Unlearnable Musical Form

To illustrate the problem that generative models struggle with learning musical form, consider a simple case: a generative model using maximum likelihood estimation optimizes the parameters θ to estimate the joint probability $p_\theta(t_1, t_2)$ from the data samples with discrete and finite values. A parametric model can estimate the joint probability if the training data samples are on a compact manifold. The data manifold can be considered compact if the combinatorial variability of the data is small relative to the amount of available training data samples. In contrast, if there is a large amount of variation in t_1, relative the number of training samples, then $p_\theta(t_1, t_2)$ is reduced to $p_\theta(t_2)$ as $p(t_1)$ becomes virtually uniform. With high-dimensional data, the problem of a non-compact data manifold is increased. Due to the curse of dimensionality, an exponentially larger amount of data is required to preserve the density of the samples in the

space. In a diverse musical dataset, many parameters vary across music pieces even when they share the same musical form. The combinatorial variability over long time periods is so large that even simple musical forms become extremely difficult for generative models to learn.

Figure 4.1. Illustrating the incoherence in images generated by Dall-E 3 (https://chatgpt.com/), Midjourney (https://midjourney.com/), and Meta AI (https://meta.ai/). These inconsistencies are evident in images featuring mirrors and wavering flags. Notice the forked or merged stripes on the flags and the inconsistent reflection and incidence angles in mirrors, among the other inconsistencies.

The problem of large combinatorial variability is not unique to music data. Image generative models, such as those for drawing hands, struggle due to the extensive variability. In most training images, some fingers are occluded, leading the model to fail in learning the correct number of fingers a hand should have [70]. Furthermore, the objects that occlude the fingers can vary greatly—from fingers overlapping each other to items like coffee cups, pockets, torsos, and more—making it difficult for the model to draw partially occluded hands accurately.

This issue extends to other structures with a high degree of variation in images. Figure 4.1 illustrates two other structures (mirrors and wavering flags) with significant variation that three commercial image generators (Dall-E 3 [71], Midjourney [72], and Meta AI [73]) fail to generate coherent images for: the angle of the subject, mirrors, and points of view vary enough that the model cannot learn how a coherent reflected image should appear. Wavering flags present a similar challenge; due to the variation at large spatial scales in the training images, the model generates physically implausible images. These generated images support the argument that, as discussed, learning coherent structure in the presence of large combinatorial variability is a practical limitation.

As discussed in Chapter 3, while experimenting with diffusion models for music generation in 2022, one could consider using two diffusion models: one trained on larger structural pieces and the other on small segments. Additionally, the method proposed in [70] first generates a 3d mesh of hands based on the given text prompt. Then a diffusion 2d image generator conditioned on the 3d hand mesh generates the final 2d image with the hands. The authors compare the proposed method against Stable Diffusion [74] fine tuned on the same training data used to train their own model. The results demonstrate that the fine-tuned Stable Diffusion model has difficulty generating accurate hand images, while their method produces flawless images of hands.

My method for generating long-form music similarly deals with large scale structures and form in a space that does not suffer from large combinatorial variability. The music generation with guidance from the structure defined in the new space can generate long-form music with coherent structure and form.

4.4 EnCodec and MusicGen

This work builds on MusicGen, a generative music model that relies on EnCodec to compress audio. This section briefly reviews these two methods. The most recent methods in the

literature for music generation follow the approach of Stable Diffusion [75]. In this approach, the generative model is trained in the latent space of an encoder. The generated vector is then decoded into audio using the corresponding decoder. The encoder and decoder are trained on a separate dataset prior to training the generative model. In the audio domain, the most commonly used models in the literature include EnCodec and SoundStream [76]. The main reason for training the generative model in the compressed latent space is to reduce the computational cost of generating long audio. For instance, EnCodec significantly reduces data size, offering a compression ratio of 150:1, which enhances the efficiency of the generative model. Some of the earlier work following this approach was based on WaveNet [77] and AutoEncoders [50].

The encoder of EnCodec is a convolution network. The architecture is similar to other common models, with residual connections, downsampling through strided convolution, and doubling the number of convolution channels whenever downsampling occurs. The convolutions blocks are followed by a two-layer long short-term memory (LSTM) to provide sequence modeling. The output of the LSTM is then passed to the last 1D convolution layer with a kernel size of 7 and 128 output channels to generate the latent vector. The architecture and parameters are configured so that the encoder produces 75 latent steps per second for 24 kHz audio, or 150 steps per second for 48 kHz audio. There are two variants of the model: a high-fidelity, non-streamable version, and a streamable version, with minor differences in their parameters. For the non-streamble model, the input is split into chunks of 1 seconds, with an overlap of 10 ms. The streamable model, after receiving 320 samples (13 ms), outputs 320 samples (13 ms).

The decoder part of the model, similar to any autoencoder, mirrors the encoder part. To upsample the feature maps in the decoder, transposed convolution is used. The decoder's output is either mono or stereo. The whole system is trained end-to-end to minimize a reconstruction loss applied over both the time and frequency domains, together with a perceptual loss in the form of discriminators operating at different resolutions.

The latent vectors are quantized for further compression. Vector quantization involves projecting an input vector onto the closest entry in a codebook of a given size. Residual

vector quantization refines this process by computing the residual after quantization and further quantizing it using a second codebook, and so forth. EnCodec has 1024 entries in each codebook, equivalent to 10 bits per codebook. The codebooks are updated during the training process and are frozen during the inference process. Multiple variants of EnCodec are trained with a varying number of codebooks. For instance, with four codebooks, the first one is used to quantize the latent vectors directly. The second codebook is used to quantize the residual or error from the first quantization, and the third and fourth codebooks are used to quantize the residuals from the previous quantization. To convert the discrete representation back to a vector, the corresponding codebook entries are summed before going into the decoder.

In MusicGen, an autoregressive model is trained in the quantized latent space of EnCodec to model music. Two variants of MusicGen, one with a 10-second and another with a 30-second context window, are trained. In my experiments, the variant with the 30-second context window is used. Given that the latent vectors are quantized, the problem of modeling sequences of them as sequences of tokens can be treated similarly to modeling sequences of characters in text. Therefore, MusicGen, similar to a natural language model, learns from sequences of tokens. The only difficulty is that each one of EnCodec's vectors is quantized into multiple tokens, for instance, four. The second token depends on the first token, as it is computed from the quantization error of the first token.

In MusicGen, a general approach is proposed to generate the tokens, experimenting with multiple patterns of prediction. Both subjective and objective evaluations show that the exact and slow flattening patterns outperform the others. The second-best performing pattern in terms of generated music quality is the delay pattern, which predicts four tokens at each step. In the delay pattern, at step t, the model predicts the token from the first codebook at t, the token from the second codebook at t-1, the token from the third codebook at t-2, and so on.

To condition the model on text, multiple text encoders are evaluated, with T5 [54] shown to have the best subjective and objective music quality in the experiments. During training, a random subset of 20% of the iterations is selected, and for these iterations, the text condition is

omitted by replacing the text embedding vector with a vector containing a specific value. During inference, classifier-free guidance (CFG) is used to generate samples with text conditioning. In CFG, token probability distributions are first predicted without text conditioning, followed by a second step where predictions are made with text conditioning. These two distributions are then interpolated at each step, and a sample is drawn from the resulting interpolated distribution.

MusicGen is trained on 20K hours of music. Half of the training data is private and internal at Meta. The other 10K hours of training music are taken from ShutterStock (https://www.shutterstock.com/music) and Pond5 (https://www.pond5.com) vocals-free music data collections.

4.5 Large Language Models

The transformer based language models are trained in two phases. In the first phase, or the pretraining phase, a large text dataset is used to train the model to predict the next token. For instance, Llama 2 [78] is trained on about 2 trillion tokens of data. After pretraining, the model can complete a given text, but is considered bad at any other task, including answering questions and following instructions. To use a pretrained language model, to answer math questions or reason, the input text prompt is crafted such that the following text contains the answer to the question. Training the model to predict tokens of the training data may be the reason these models memorize the training data. [79] show that some random subset of the training data is memorized by LLMs. And larger models memorize a larger portion of the training data.

The second training phase fine-tunes the pretrained model to simplify the use of the model for a specific task. For instance, Llama 2-Chat is based on Llama 2 trained in the second phase to be used as a general AI agent answering questions, following instruction and so on. This fine-tuning phase uses a much smaller dataset, often with high-quality samples for the model to learn from. For instance, for the instruction-following models, the fine-tuning dataset contains instruction and answer pairs. While the model is given all the tokens from each instruction and

answer pair, the backpropagation process is only applied to the tokens of the answers. The model after the second training phase is also called an aligned model, as it is expected to be aligned with human expectations for the task.

To use an aligned model for a narrow use case, for instance proposing prompts for a text-to-music model, often a third process is needed. The two main approaches are in-context learning and fine-tuning. It is proposed by [59] that a prompt structure be composed of a task description followed by several canonical examples. Through this prompt, the model is able to learn how to respond to an instruction without requiring updates to its weights. It is not known how transformers learn from in-context samples. In the literature some possible explanations have been proposed, for instance [80]. An advantage of the in-context learning approach is that iterating on prompt design is quick. Another advantage is that rather than requiring a separate copy of the model for each task, a single model can be used for several narrow tasks using in-context learning.

Therefore, the prompt used for in-context learning has an important role in the success of this approach. Several efforts have been reported in the literature to improve the prompt design process. Shin et al. [81] frame the prompt design process as a discrete space search problem and show that this method can design prompts that outperform manual prompts. Lester et al. [82] propose "soft prompt", a method that uses the task examples to add tunable tokens to a prompt to improve the performance of the prompt for the given task. In this chapter, I use in-context learning in the method as discussed in the following sections. A later section also discusses my proposed method for automatic prompt design.

In the fine-tuning approach to adapt an aligned model for narrow tasks, unlike in-context learning, the weights of the model are modified. Given the large number of parameters in LLMs, updating the weights using a few samples is not expected to be effective. Hu et al. [83] propose a method, LoRA, that is based on the hypothesis that LLMs have a low intrinsic dimension. Therefore, in LoRA, a small number of parameters are trained using a low-rank decomposition of the update weight matrix. There are other similar approaches in the literature to reduce the

number of trainable parameters, including IA3, which trains a vector in the attention layers [84] and FedPara that learn low rank matrices similar to LoRA but is based on the sum of multiple matrices [85].

Earlier LLMs had a small context window, limiting the length of the task description and examples for in-context learning that could be passed to the model. For instance, Llama's context window length was 2,048 tokens in the first generation, while it is 8,192 tokens in the third generation. Google's Gemini 1.5 Pro, released in 2024, can process up to two million tokens, which is approximately 1.2 to 1.6 million words. The context window of ChatGPT 4 is 128k tokens long. The increased context size has been shown to improve in-context learning performance compared to fine-tuning when providing a large language model (LLM) with a large set of examples, up to around 1,000, for a given task [86, 87].

4.6 Controlling MusicGen by a Language Model

As MusicGen is conditioned on text, it affords an interface in natural language. Thus, an LLM can generate prompts for MusicGen to replace human prompts. Therefore, it is possible to task an LLM with designing the structure of a song and create prompts for each section of the music to be generated by a text-to-music model. This capability of an LLM to design music structure and generate prompts is supported by its diverse knowledge base [88], the reasoning abilities [89], and learning capabilities [59].

In this work, ChatGPT 4 is used as the LLM, which is GPT 4 [58] fine-tuned to answer questions. One challenge is aligning the LLM with the text-to-music model. MusicGen has been trained on brief music descriptions that are not technical and adhere to a certain style. In this work, few-shot in-context learning is applied to instruct ChatGPT to generate prompts for MusicGen by providing 50 song descriptions from Pond5. With enough examples, the music generated by MusicGen accurately reflects the text prompt. My empirical results show that when using about 10 or fewer song descriptions (zero-shot or one-shot learning), ChatGPT's generated

prompts were often misinterpreted by MusicGen. Increasing the number of song descriptions in the prompt from 50 to 80 did not improve the interpretability of ChatGPT's generated prompts by MusicGen. As LLMs are prone to hallucination, [90], using more samples than necessary should be avoided.

Asking ChatGPT to first come up with a musical form and then write prompts for each part aids in achieving diversity in the musical forms. The instruction prompt also specifies that the length of each piece is 2.5 minutes. The instruction prompt to ChatGPT is presented in Listing 4.1. To summarize, the prompt has four sections. First, the task is defined for the model. In my experiments, the main prompting strategy that significantly improves the quality of the generated responses is the chain of thought approach. That is the reason, the instructions ask ChatGPT to first design the form, then justify the role of each part, and finally write the prompt for each part. The second section of the prompt to ChatGPT provides some sample music descriptions for in-context learning, which are taken from Pond5. A set of rules is provided in the third section, including the minimum and maximum length of each section. The last section instructs ChatGPT to generate its response in a structured format, JavaScript Object Notation (JSON). My Python script parses the response from the ChatGPT API and controls the MusicGen to generate the audio. A sample response from ChatGPT is presented in Listing 4.2.

The LLM is responsible for generating prompts for each section of the music piece. However, generating the audio for each section individually using MusicGen and then joining them together results in abrupt transitions. To address this, I use a technique similar to the CFG method. Instead of calculating a single text-prompt-conditioned probability distribution, two distributions are estimated based on separate text prompts. The interpolated distribution gradually shifts the weight from the first section's prompt to the second, decreasing from one to zero for the first and increasing from zero to one for the second over five seconds. This approach enables the model to generate smooth and continuous transitions between sections.

In a similar fashion, to create sections that vary from earlier parts—such as producing section A' by referencing section A—in addition to the extra text-prompt conditioned probability

distribution, a 15-second audio prompt from the end of section A is provided to MusicGen to predict one more audio-only-conditioned probability distribution. Before applying the CFG process, first the two text-prompt-conditioned, and then the two audio-only-conditioned distributions are interpolated. The final two distributions, one audio-only and one text conditioned, are then passed through the CFG process, resulting in smooth and seamless transitions between sections in practice.

LISTING 1: Prompt to ChatGPT

```
Assume you're a musician. You can write text prompts for a system
that generates the music for the given description of the music. The
    system was trained with description from a stock music catalog,
    descriptions that will work best should include some level of details
    on the instruments present, along with some intended use case (e.g.
    adding "perfect for a commercial" can somehow help).
The following are some example prompts that the system understands and
    the music it can generate:
- "A cheerful country song with acoustic guitars"
- "Lofi slow bpm electro chill with organic samples"
- "Electro swing"
- "90s rock song with electric guitar and heavy drums"
- "a light and cheerly EDM track, with syncopated drums, airy pads, and
    strong emotions bpm: 130"
- "Chillstep, calm EDM"
- "A smooth jazz fusion piece blending electric piano, a stand-up bass,
    and a soft trumpet solo. Perfect for a sophisticated lounge setting"
- "A relaxing tropical house track with steel drums, a gentle electronic
    beat, and a breezy melody, great for a summer beach party scene"
```

- "Complextro"
- "Smooth guitar ballad"
- "An ambient music piece with ethereal synth pads, a gentle piano melody, and subtle nature sounds, creating a serene and mystical atmosphere"
- "Japanese traditional music koto"
- "Japanese traditional music flute and koto"
- "J-Pop, 50bpm" - "A classic soul track with a warm bass line, smooth electric guitar, and soulful vocals, capturing the essence of a 70s soul club."
- "A lively Latin track with vibrant brass, energetic percussion, and catchy piano riffs, suitable for a festive party or dance scene"
- "A nostalgic 80s synthwave track with vintage synthesizers, a driving bass line, and electronic drums, perfect for a retro-themed project or game."
- "A sweeping orchestral composition with soaring strings, powerful brass, and dramatic percussion, ideal for an epic movie trailer"

Don’t limit yourself to these sample prompts. Be creative and compose music piece by writing prompts for each part. Such that after stitching the parts together an impressive music piece is generated that most people would find creative and enjoyable. The music piece should be coherent and have unity. Write the thought process for the composition, followed by the parts. Don’t change the music style often so the piece is unified and holds cohesion. Consider complex music form for the piece and describe the form first. Very important constraints that your prompts have to satisfy: 1. The whole piece should be exactly **150 seconds**. You also decide how long each part should be. 2. The prompt of each part can reference another part.

```
REFERENCED_PART is either the number of a previous part or -1 meaning
no other part is referenced. 3. Consider David Huron's ITPRA framework
 when planning the music form. 4. Do not use comparative adjectives in
 the prompts. For example, do not say a "faster tempo than the
previous part". Instead specify the BPM values. 5. Do not fuse music
genres. 6. avoid slow or soft music. 7. keep each part no longer than
45 seconds and no shorter than 20 seconds. 8. Use the form to make the
 music engaging and interesting, as each part by itself is not that
interesting. 9. Avoid world fusion music. 10. Write the form of each
piece first. 11. DO NOT use vague language. Be verbose in the prompts.
 For example if want more layers in one part, just write exactly what
instruments should be used as layers. 12. Create contrast through BPM,
 texture, and instruments between parts. 13. avoid comparative phrases
 like "added layers" and "fuller sound". Instead be verbose, be
precise and add absolute phrases only. 14. To repeat a part with
variation in the chosen music form, reference that original part and
in the prompt for the new part mention what changed. Write the final
output in the JSON code format like this: { PART_NUMBER: ["PROMPT",
LENGTH_IN_SECONDS, REFERENCED_PART], PART_NUMBER: ["PROMPT",
LENGTH_IN_SECONDS, REFERENCED_PART], ... } Create 10 such pieces. Each
 of the 10 descriptions is followed by the corresponding JSON snippet.
```

LISTING 2: A sample response from ChatGPT

```
Music Piece 3: "Electro Dance Journey"
Form: A-B-C-B'-A'-C'
Part A: Introduction of the main dance theme
```

```
Part B: Development with new elements
Part C: Bridge introducing contrast
Part B': Return of B with variations
Part A': Return of A with changes
Part C': Final contrasting section with a climax.
Prompts and Structure:{ "1": ["An energetic electro dance track with a
    driving beat, synth leads, and rhythmic bass. BPM: 130", 25, -1], "2":
     ["A lively electro dance part with additional staccato synth chords
    and a more complex drum pattern. BPM: 132", 25, -1], "3": ["A
    contrasting electro dance bridge with a softer tone, featuring melodic
     synth lines and a slower beat. BPM: 125", 20, -1], "4": ["Return of
    the upbeat electro dance theme from Part 2, now with a richer
    arrangement including layered synths. BPM: 132", 30, 2], "5": ["
    Variation of the initial electro dance theme, integrating elements
    from previous parts for a fuller sound. BPM: 130", 25, 1], "6": ["A
    climactic electro dance finale, combining the energy of all parts with
     increased tempo and intensity. BPM: 135", 25, 3]}
```

4.7 Evaluation

In this section, I introduce a novel objective metric designed to evaluate the musical form of generated music. This metric not only allows for the evaluation of my method and MusicGen but also assists in refining parameters, including the mega prompt, to better align the generated musical forms with those in pieces from Pond5. Objective evaluation results demonstrate that my method produces music with a more diverse and complex range of forms compared to MusicGen. Additionally, subjective evaluation results are presented, supporting this finding.

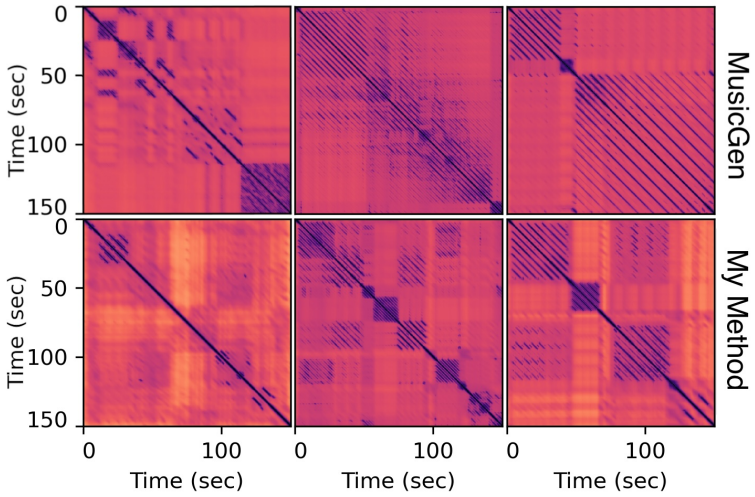


Figure 4.2. Visualizing the fused self similarity matrices [2] of three songs generated by MusicGen, left column, and three songs generated by my method, right column. Comparing the two columns, it is evident my method can generate songs with repeating parts and more variations over the whole song. A common colormap range is used in all the heatmaps. The songs generated by MusicGen (top to bottom): 1, 2, 3; the songs generated by my method: 1, 2, 3.

4.7.1 Objective Evaluation

The use of self-similarity matrices (SSMs) to visualize and analyze music structure dates back to at least 1999, when Foote introduced a similarity metric along with various applications for SSMs [91]. Since then, SSMs have been widely applied, including in measuring structural similarity between pairs of music pieces [92] and in searching for specific structural matches within a set of music [93], among other uses.

In my work, a metric that solely analyzes the structure of individual music pieces or compares structures between pairs is inadequate. The generative method I propose aims to consistently produce a set of music pieces with structural characteristics akin to human-composed works. This objective boils down to a primary requirement: the structural distributions of the generated and human-composed music sets should be similar. To my knowledge, no evaluation metric addressing this need for musical structure has yet been proposed in the literature.

For such an objective evaluation metric, the similarity matrix must incorporate multiple

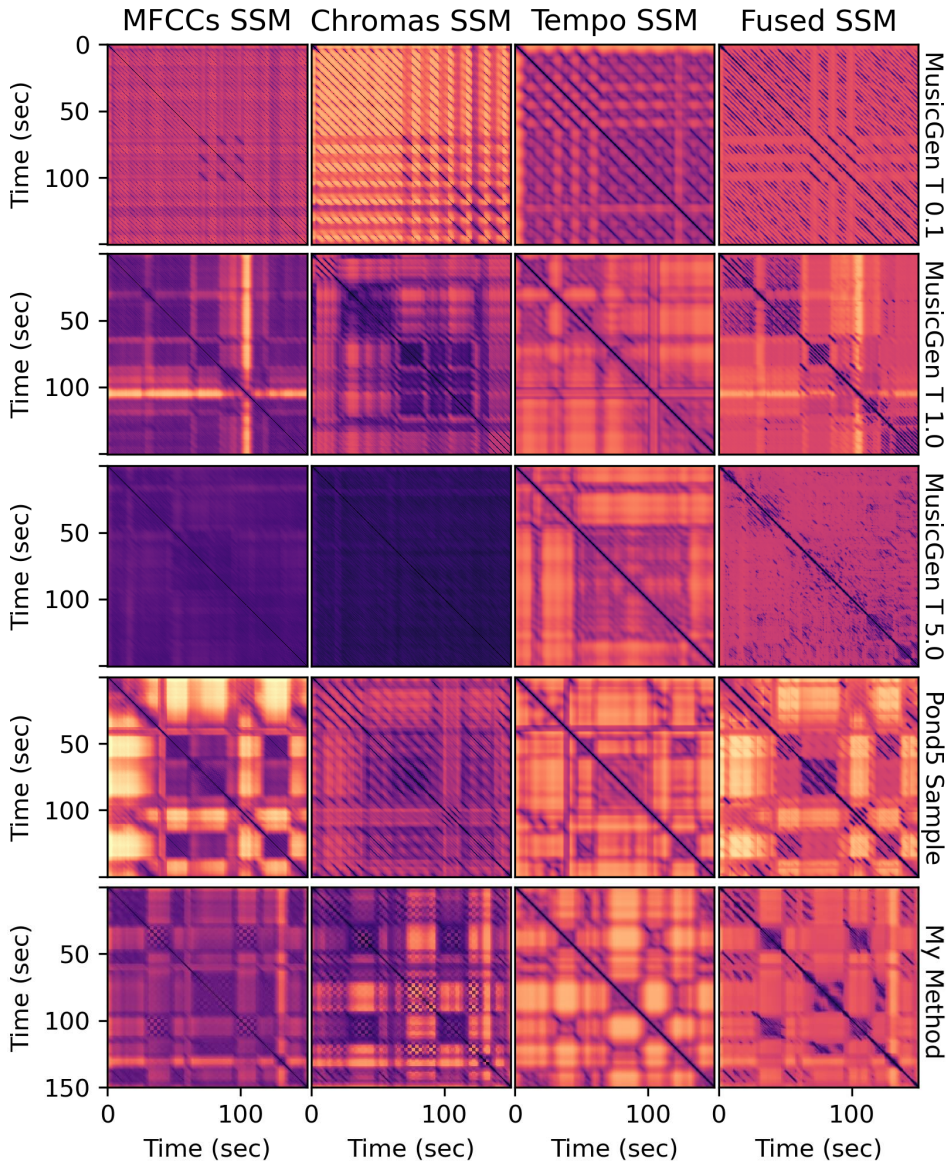


Figure 4.3. Visualizing the self similarity matrices for 3 MusicGen samples, one sample from my method, and one from Pond5. With MusicGen, at a low temperature (T) of 0.1, the music is repetitive. At T=5.0, there is mostly random noise. At T=1, the music is meandering. The sample from my method resembles the one from Pond5, composed and arranged by a musician.

musical metrics—beyond just chroma or tempo—to fully capture the musical structure. The similarity network fusion (SNF) method proposed by Tralie and McFee [2] has demonstrated superior noise suppression in combined SSMs compared to other methods. Thus, I use the SNF method to extract a single SSM that represents the musical structure of each piece. To

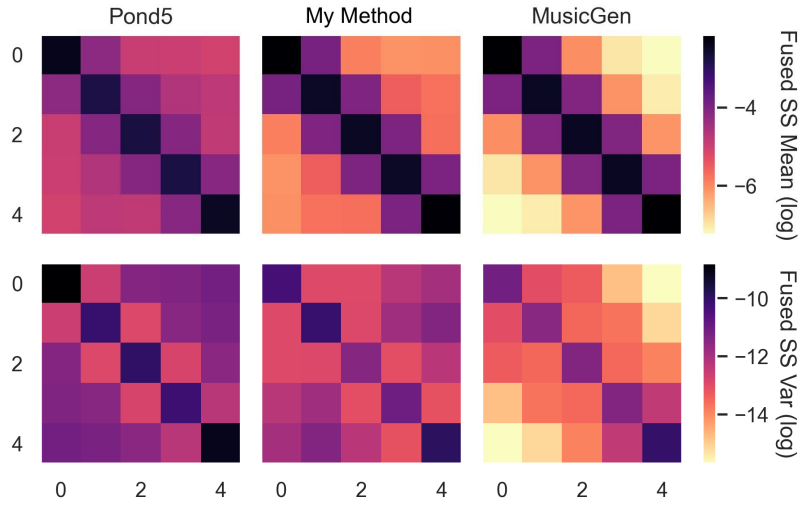


Figure 4.4. The mean (top row) and the variance (bottom row) of the fused self-similarity (SS) matrices are estimated with 100 samples from Pond5, generated by my method, and by MusicGen. The SS matrices are downsampled to 5×5. The results show that, compared to MusicGen, the samples from my method are more similar to the Pond5 samples in similarity of the parts at long temporal distances.

assess both the diversity of musical forms and the distribution of forms between the generated and human-composed sets, I calculate the Fréchet distance between the two sets of extracted structures.

Figure 4.3 compares the SSM of samples from MusicGen, my method, and Pond5. The SNF method proposed by Tralie and McFee [2] is used to generate the combined SSMs. Increasing the temperature parameter of the sampling method, from 1 to 5, does not lead to more variation in the music structure generated by MusicGen but leads to noisy audio. Reducing the temperature value to 0.1 leads to consistently repetitive output by MusicGen. At the default temperature value of 1, the generated music meanders. The structure of the samples generated by my method resembles the structure of the samples from Pond5 in the amount of variation and similar parts.

Figure 4.2 compares the fused SS matrices of three samples generated by my method and by MusicGen. To remove the influence of varying musical material and audio quality, the first 10 seconds of audio from my method's samples were used as an audio prompt for MusicGen. The

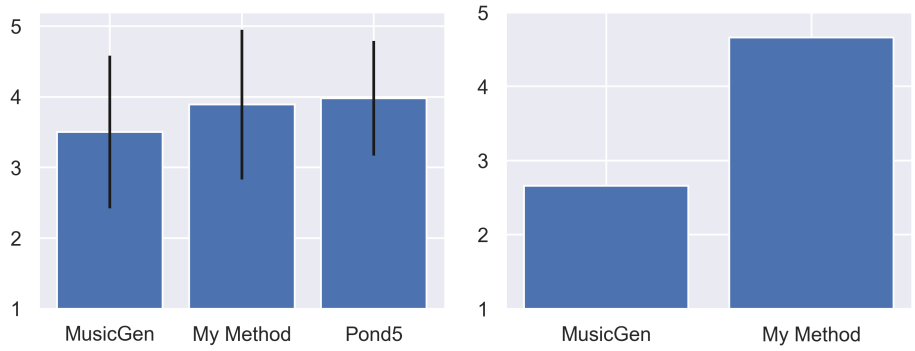


Figure 4.5. Left: The subjective comparison of the generated music and sampled from Pond5 by non-musicians is measured through the MOS, based on how engaging the music is. Right: the subjective comparison of the samples by musicians, critiquing the musical structures.

caption of Figure 4.2 contains links to the generated audio samples, which you can click on to listen.

After downsampling the fused SS matrices of 100 samples, the mean and variance matrices are estimated for Pond5, my method, and MusicGen in Figure 4.4. Therefore, this figure helps compare the distributions of the music structures. It is apparent, from the mean and variance values in the top-right of the matrices, that the MusicGen samples do not have parts near the end of each piece that resemble the parts near the beginning. In contrast, the samples from my method are more similar to the samples from Pond5. The Fréchet distance between the two pairs of distributions is estimated, but instead of using Inception feature vectors [94], the mean and covariance matrices of the upper triangle of the fused SS matrices are used. The Fréchet distance between the distributions of the Pond5 samples and my method's samples is 0.086, and between the Pond5 and MusicGen samples is 0.108, supporting the claim that the structure of the samples generated by my method is more similar to the samples composed by musicians from Pond5.

4.7.2 Subjective Evaluation

For a subjective evaluation by non-musicians, 10 samples using my method are generated, 10 samples using MusicGen, and 10 samples from Pond5 are selected. Each of these samples is 2.5 minutes long. Using Amazon Mechanical Turk (MTurk), a mean opinion score (MOS) between 1 and 5 for each sample is estimated, the average of 10 scores from non-musician subjects. The human evaluators are asked to evaluate the overall quality of the music, following the recommended practices in CrowdMOS [95]. The subjects are told that a score of 1 means they dislike the music and would not like to listen to similar music. A score of 5 means they find the music interesting and would like to listen to similar music. Given that long music tracks with more organized structure are expected to be more engaging, the likability of the samples is used as a proxy for improved structure and form. The results in Figure 4.5 (left) indicate that adding musical form through my method to MusicGen improves the perceived quality of the music, almost on par with human-composed music pieces from Pond5.

In a separate subjective evaluation, the three subjects selected for this study were professional musicians with formal music training, each possessing at least five years of performance experience. They were asked to listen to three samples from my method and three samples from MusicGen, which are the same samples referenced in the caption of Figure 4.2. The songs generated by MusicGen are available at: 1, 2, 3, and the songs generated by my method are available at: 1, 2, 3. The subjects rated each sample using the following scoring guideline: 1: No form, music meanders; 2: Minimal structure, some recognizable patterns but largely unstructured; 3: Moderate structure, clear sections but not highly organized; 4: Clear form, well-organized sections and transitions; 5: Very clear and highly structured, strongly organized and cohesive. The average scores are shown in Figure 4.5 (right).

4.8 Meta Optimization by LLM

In the previous section, the prompt provided to the LLM for in-context learning is designed manually following a tedious trial-and-error process. LLMs, while they understand natural language, their interpretation of the input text is sensitive to phrases and words. For instance, a chain of thought that asks the LLM to solve problems in multiple steps [96] can change the responses of some LLMs. The nuances in the input prompt then motivate exploring the prompt space for a specific task, such as generating prompts for generative music models. There are three downsides to exploring the prompt space manually. First, coming up with a new prompt to evaluate is a daunting and tedious task. Second, in practice, only a small space can be explored. Third, LLMs such as GPT are regularly updated, and their responses to the same prompt can change, requiring the prompt engineering process to be repeated.

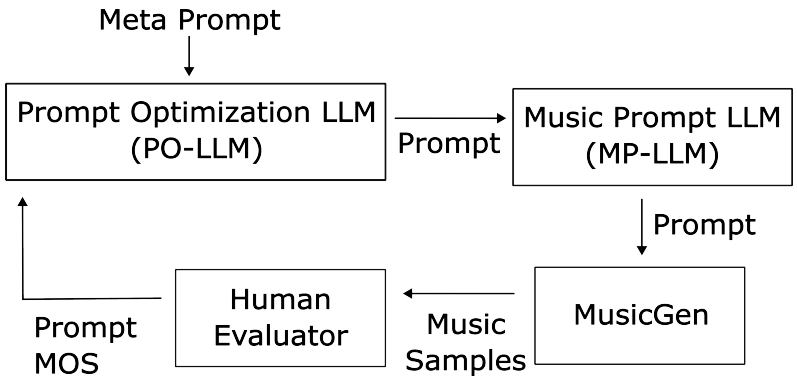


Figure 4.6. The PO-LLM proposes new instructions and few-shot samples for the MP-LLM. The MP-LLM follows the instructions and generates a set of prompts for MusicGen to generate a coherent music piece with a musical form. The generated music is then rated by human evaluators, and the average MOS is estimated. The PO-LLM is instructed by the meta prompt to consider the previous 5 prompts with the highest MOS to propose a new prompt for the MP-LLM.

This section proposes an automatic process to optimize the prompt. The high-level process is illustrated in Figure 4.6. There are two LLMs in this process. The Music Prompt LLM (MP-LLM), as in the previous sections, generates prompts for the generative music model. The Prompt Optimization LLM (PO-LLM) generates new prompts for the MP-LLM. The PO-LLM

and MP-LLM are both ChatGPT 4 in my experiments, in two separate sessions. However, in general, these could be two different LLMs. For the optimization, a scoring function is also needed. Here, the MOS of the generated music by the generative music model is used as the scoring function. Therefore, the PO-LLM attempts to maximize the MOS of the generated music by optimizing the instruction prompt for the MP-LLM, which in turn generates the prompts for the model generating the music.

It has been shown that LLMs can be used as black-box optimizers, for instance, to optimize the coefficients of a linear model to reduce the residual [97]. The optimization problem here is expected to be harder because, instead of optimizing the input to a single model, two stochastic models are chained together, and the subjective score is noisy. My experimental results are promising despite the challenging setup of the optimization problem. The crucial piece is the meta prompt, the prompt that explains the optimization problem to and instructs the PO-LLM.

In the experiments, two separate meta prompts are used, one for each phase of the optimization process. In the exploration phase, from a single seed prompt, which is the best hand engineered prompt, a diverse set of 20 prompts is generated by the PO-LLM. Each of the generated prompts is then used to generate 10 music pieces, and each piece is scored by five subjects. The exploration meta prompt is presented in Listing 4.3, and the prompt from the previous section is passed as the sample.

LISTING 3: Meta prompt in the exploration phase

```
Assume you are a music composer who is creative and composes in several
    genres. You want to use a large language model (LLM) to come up with a
     new composition. The LLM writes a high-level description of each part
     in the piece. You write a set of prompts for the LLM. A sample prompt
     is provided to you. You should use this sample to get inspired, do
    not limit yourself to this, and do your best to be creative to come up
```

```
 with a novel and diverse set of instructions. Write 20 separate
prompts for the LLM in the JSON format ['_prompt_1','_prompt_2_',..].
Keep the prompts concise. The sample prompt is:
```

To begin the exploitation phase, five prompts with a diverse range of MOS are taken from the prompts generated in the exploration phase and passed to the PO-LLM. In each iteration, a new prompt is proposed by the PO-LLM and evaluated. The new prompt is added to the set of top five prompts if its score is higher than the lowest score in the set. In each iteration, only the top five prompts are retained. After 20 iterations, the optimization process is stopped. The meta prompt in Listing 4.4 is passed to the PO-LLM along with the pairs of prompts and corresponding scores:

LISTING 4: Meta prompt in the exploitation phase

```
Assume you are a music composer who is creative and composes in several
   genres. You want to use a large language model (LLM) to come up with a
    new composition. The LLM writes a high-level description of each part
    in the piece. You are given five such prompts with their
   corresponding scores. You should think of a new prompt that is going
   to have a higher score than the five samples. The samples projects and
    their scores are provided in the JSON format: [{'prompt':'_prompt_1
   ','score':_score_1}, {'prompt':'_prompt_2','score':_score_2}, ..]
```

LLMs are more prone to hallucination when provided with a larger prompt [90]. As the meta prompt can be very large, providing the list of prompt and score pairs in a format such as JSON can help reduce the risk of hallucination. It has been claimed by some studies that the HTML format reduces the chance of hallucination more than the other formats for lists passed to

LLMs. This is probably due to the training data of most LLMs containing a large number of HTML documents. In my experiments, the JSON format is used, which keeps the prompt easy to read and well-structured for the LLM, without increasing the hallucination rate.

Figure 4.7 presents the average MOS of the generated music by the prompts generated in the exploration phase. The majority of the scores are lower than the score of the initial prompt, 3.88. The lower scores are expected, as the initial prompt was tuned manually and probably is close to a local optimum. The spread of scores shows that the PO-LLM can generate a diverse set of prompts, which is desirable in the exploration phase.

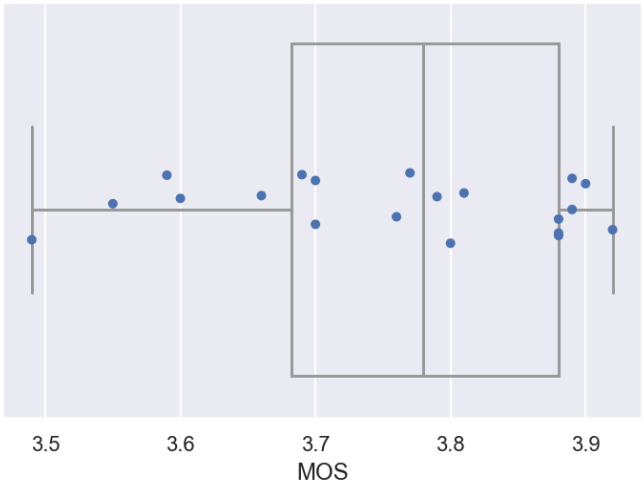


Figure 4.7. The average MOS of the prompts generated in the exploration phase of the optimization method.

The average MOS of the top five prompts over optimization iterations in the exploitation phase are presented in Figure 4.8. In more than half of the iterations, the average MOS does not change between two iterations due to the MOS of the generated prompt being lower than the top five prompts. The ascent rate and the final-to-initial average MOS ratio are above the noise level. The prompt with the highest MOS (3.93) at the last iteration is slightly better than the MOS of the initial seed prompt (3.89). The results suggest that exploring this method with a non-optimized seed prompt could be a promising area of research to fully automate the prompt engineering process.

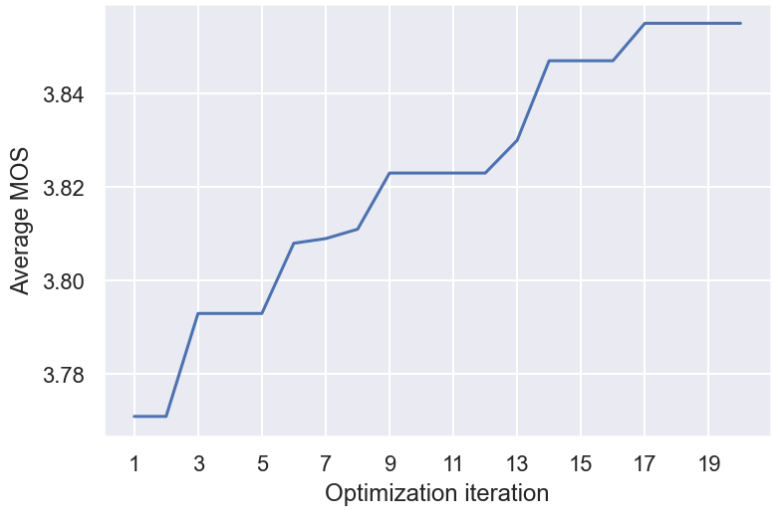


Figure 4.8. The average MOS of the top 5 prompts in each iteration of the optimization.

4.9 Conclusions

Generating short music by machine learning models has been the focus of research in the literature, with some notable progress. The generated music by such models lacks structure at the span of a minute or longer. Generating long music with structure has remained an open problem. This chapter argues that, due to the nature of the problem, the music generative models cannot learn long-scale structure or musical form from musical datasets.

This chapter presents a novel method to combine music generative models with large language models to generate music with musical form. The LLM designs the form and parts and communicates with the generative music model in natural language. The technical challenges are discussed, and the objective evaluations show that my method can generate samples with structures that resemble those in music composed by musicians. The subjective experiments also support the claim that my method generates cohesive and highly structured music compared to MusicGen.

As an extension, this chapter also presents an optimization method to optimize the prompts for in-context learning, maximizing the MOS of the generated music or some other

criterion. The limitations of the proposed methods are discussed, which present interesting research problems for future work. For instance, continuing motives over multiple parts remains a challenge. In general, generative music models with greater control over musical aspects are an open problem. The limited size of the training data and the limited number of captured attributes within the data are the main factors. Therefore, future work on larger and richer music datasets, as well as models with control over more aspects of music, can lead to more useful tools for musicians in practice.

Chapter 5

Dimensionality Reduction

5.1 Abstract

This chapter reviews the literature on some of the dimensionality reduction methods and their usage in music interfaces. In the first section, the motivation for using dimensionality reduction methods in digital musical instruments is presented. The second section covers some of the methods used for dimensionality reduction, such as PCA, LDA, SNE, t-SNE, Hinged-t-SNE, parametric t-SNE, UMAP, Autoencoders, and VAEs. Some of my own experiments are also discussed in the third section. The fourth section reviews approaches to using some of these algorithms in musical interfaces. My thoughts on interesting research problems and new research directions are presented in the last section.

5.2 Introduction

Digital musical instruments (DMIs) have become more commonly used by practitioners since the design of digital synthesizers. The novelty and flexibility of DMIs have attracted researchers to explore several aspects of them. Particularly, novel interfaces for DMIs are actively being researched. A prominent conference where researchers share their work is the International Conference on New Interfaces for Musical Expression, also known as NIME.

Miranda and Wanderley in [98] define digital musical instruments as "an instrument that uses computer-generated sound . . . and consists of a control surface or gestural controller,

which drives the musical parameters of a sound synthesizer in real time". The results of the survey from a diverse group of practitioners show that software is the most common digital musical instrument, followed by MIDI controllers [99]. The authors explain that the reason for the popularity of software is its flexibility and portability. Hardware interfaces are commonly used to control software. The authors in [100] note 2d touch interfaces are a popular type of interface by practitioners. The authors survey some practitioners trying to find what 2d touch interfaces are commonly used by practitioners to control DMIs. The paper claims iPad is the most common touch interface. The flexibility and portability of the iPad are also used to explain its acceptance among practitioners. A common 2d controller consists of a rectangular surface, with the coordinates of a point within it serving as the control parameters. And the 2d coordinates are typically mapped to control multiple parameters of a synthesizer at the same time.

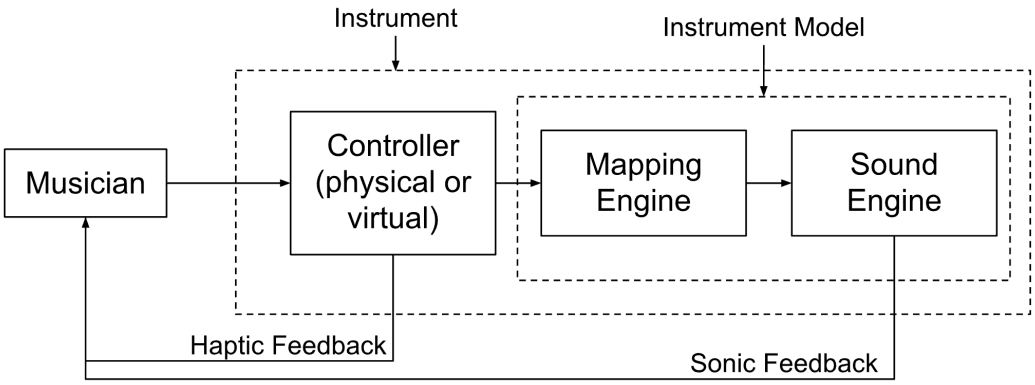


Figure 5.1. Depicting the components of a digital musical instrument. The mapping engine and sound engine together are considered the core of the instrument and define the characteristics of the instrument. Adopted from [3].

Magnusson [3] proposes breaking down a DMI into three separate components. The controller is the component the musician interacts with directly, such as a slider or a 2d area on a touch screen. The sound engine is responsible for generating the sound of the instrument. The mapping engine maps the output of the controller to the parameters of the sound engine. Figure 5.1 shows these components and the connection between them. Magnusson [3] argues in all DMIs, the controller or interface can be exchanged for a controller with similar affordances

without much change in musical expression. However, the constraints that define the characteristics of a DMI are set in the mapping engine and sound engine. Therefore, together, the mapping engine and sound engine are the core of a DMI.

With few control parameters and few sound engine parameters, the instrument designer can decide on the mapping. For instance, in my previous research [101], the mapping from tracked body points in a camera are tied to the parameters of the synthesizer. Designing mappings with many control parameters or many sound engine parameters can become a tedious task or even impractically difficult. An approach used by instrument designers is to use dimensionality reduction methods to reduce the number of dimensions that need to be mapped. With a reduced number of dimensions, the designer can more easily consider the constraints and expressiveness of the mapping. The focus of this paper is on dimensionality reduction methods for this purpose.

Dimensionality reduction is the process of reducing the number of features to the most relevant ones, in simple terms. Relevance or importance is defined by the objective that the algorithms aim to minimize or maximize. In general, the downside of using dimensionality reduction is that some information is lost along with the noise in the input signal. Dimensionality reduction methods are mostly used for noise reduction, data compression, data clustering, and data visualization.

There are two general approaches to dimensionality reduction. The first approach is projection, which involves projecting points in the high dimension to a lower dimension while preserving the distances between points in the higher dimension. The second approach is manifold learning, a non-linear method based on the manifold hypothesis, which posits that most real-world high-dimensional datasets lie close to a much lower-dimensional manifold [102]. Some of the most commonly used dimensionality reduction algorithms include PCA, t-SNE, LDA and UMAP. In the next section, the properties of these algorithms are discussed.

Dimensionality reduction methods are mainly designed and used to alleviate the curse of dimensionality. The curse of dimensionality refers to the counter-intuitive sparsity of the high dimensional spaces. This means that if a set of random points are uniformly placed inside a unit

hypercube, as the dimensionality increases, the distance between points grows larger. At a high number of dimensions, the points are almost all at the edges of the hypercube. To observe this phenomenon, another hypercube with a side length of 0.9 is situated inside the unit hypercube. In one dimension, this inner hypercube contains 0.9 of the uniformly distributed points, in two dimensions $0.81 = 0.9^2$, and in n dimensions 0.9^n. Therefore, in the 50 dimensional space, the inner hypercube with side length that is 90% of the unit hypercube side length contains about 0.5% of the points, and the other 95.5% of the points are outside of the inner space. As a consequence, in high dimensional spaces the distance between the points cannot be reliably used to find similar points because the similar points are likely to be as far as dissimilar points. To alleviate the curse of dimensionality for algorithms that use some concept of distance between data points, the data is preprocessed as follows: First, a dimensionality reduction algorithm is used to bring the data points to a lower-dimensional space. Then, the algorithm that relies on the distance between points is applied to the points in the lower-dimensional space.

In this chapter, the second section will cover some of the dimensionality reduction methods. Some of my own experiments are also discussed in the third section. The fourth section reviews the approaches to using some of these algorithms in musical interfaces. My thoughts on interesting research problems and new research directions are presented in the fifth section.

5.3 Dimensionality reduction algorithms

The dimensionality reduction methods can be divided into linear and non-linear methods. In linear methods, the data points are projected onto a lower dimensional linear manifold. In non-linear methods, the lower dimensional manifold is non-linear.

5.3.1 Linear methods

Linear methods are generally faster than non-linear methods. It is also easier to interpret the output of linear methods, as the points are mapped to a linear manifold. PCA and LDA are two well-known linear dimensionality reduction methods.

Principal Component Analysis

Principal Component Analysis (PCA) in [103] is a linear method for dimensionality reduction that has been used in a wide range of applications. It is an unsupervised algorithm since it does not require classes labels and focuses on finding the principal components that maximize the variance of the mapped data. Principal Component: the axis that shows the variance among the data in a training set. In a two-dimensional space, two principal components are present, which are orthogonal to each other. The higher the dimensionality, the more orthogonal axes (principal component vectors) there will be. For visualization purposes, it is common to limit the representation to only two or three dimensions.

The typical method to compute PCA is based on singular value decomposition (SVD). An SVD of a real $m \times n$ matrix A of rank r is a factorization $A = U\Sigma V^\top$ where, U is an $m \times r$ matrix such that $U^\top U = I_r$, V is an $n \times r$ matrix such that $V^\top V = I_r$, and Σ is an $r \times r$ diagonal matrix with the diagonal elements $\sigma_1 \geq \cdots \geq \sigma_r$ are strictly positive and are called the singular values of A. To apply PCA to a set of points $x_1, ..., x_n$ in $\mathbb{R}^m$, first the mean data point is calculated $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$. Then the data points are mean centered by replacing x_i by $x_i - \mu$. The mean-centered data points are then placed as the columns of a $m \times n$ matrix X. The columns of $Z = U_k^\top X$ are the data points in the lower dimensional space k. The columns of U_k are the vector corresponding to the k singular values in the SVD factorization of X.

There are several variants of PCA that add some desirable property to regular PCA. For instance, sparse PCA [104] finds the principal components that increase variance and also the sparsity of the projected vectors, i.e., there is a small number of non-zero values. A limitation of PCA is that in order to compute the SVD factorization of the data matrix, all the data points should be loaded in computer memory. For a large dataset, the matrix might be too large to fit in memory. The incremental PCA [105] addresses this issue by using subsets of the data points in batches to compute the principal components.

Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) [106] is a supervised algorithm. It computes the linear discriminants, which maximize the separation between multiple classes. It is mostly used in the pre-processing step for pattern-classification for avoiding overfitting and lowering the computational cost by projecting a high-dimensional space data-set into a lower-dimensional space with a class separability. LDA tries to find the axes that maximize the separation between multiple classes. Whereas PCA tries to find the axes that will maximize the variance in the data set. LDA and PCA have similar approaches, but usually LDA is used after PCA as it is proven that PCA works better than LDA alone as a projection algorithm for dimensionality reduction [107].

To compute LDA, the dimensional mean vectors for each class in the data set are computed. Then, the between-class S_B and within-class scatter S_W matrices are computed. Finally, the K eigenvectors of $S_W^{-1}S_B$ with the largest corresponding eigenvalues are used to map the data to a K-dimensional space.

Supervised dimensionality reduction methods such as LDA are not as useful as unsupervised methods for musical interfaces. This is because assigning two or more labels does not make much sense in this application. For example, it is not straightforward to assign a set of synthesizer parameter values to multiple classes.

5.3.2 Non-linear methods

Non-linear methods map the given data points to a non-linear manifold. These methods are designed to preserve more of the local distances at the cost of loss of global structure when mapping to a lower dimensional space. Some of the common methods in the literature are reviewed in this section.

Stochastic Neighbor Embedding

Linear dimensionality reduction methods, like PCA (Principal Component Analysis), keep far points in the high dimensional space far in the low dimensional space [4]. However, they might map close points in the high dimensional space to points that are far in the low dimensional space. Therefore, the local structure is not preserved by linear dimensionality reduction methods. PCA, a linear model, cannot keep similar points in a high dimensional space near each other after mapping the points onto a two dimensional space. Algorithms such as Stochastic Neighbor Embedding (SNE) and t-distributed Stochastic Neighbor Embedding (t-SNE) attempt to preserve the local structures when mapping high dimensional data to a lower dimension data.

In a high-dimensional space let's have a data set:

$$X = \{x_1, x_2, \ldots, x_n\} \tag{5.1}$$

In a lower dimensional space (two or three dimensions) let's have data set

$$\mathscr{Y} = \{y_1, y_2, \ldots, y_n\} \tag{5.2}$$

The set $\mathscr{Y}$ is referred to as a map. The individual points y_i are referred to as map points.

A probabilistic measure is defined to measure how close two points are to each other, considering the distance to the other points as well. Using a normal distribution that is centered at point x_i the probability of x_j being drawn from this distribution is calculated in the numerator of

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2)} \tag{5.3}$$

In the denominator, the probabilities of other points being drawn from this distribution are summed. σ_i^2 is the variance of the normal distribution centered at x_i. Its value is decided based on the density of the points near x_i and a parameter that the user provides called Perplexity

(a number suggested to be between 5 and 55) [4].

$$q_{j|i} = \frac{\exp(-||y_i - y_j||^2}{\sum_{k \neq i} \exp(-||y_i - y_k||^2)} \tag{5.4}$$

Similarly, a probabilistic value measuring closeness of the points is defined for the points in the lower dimension. The only difference is that the variance of the normal distribution is set to $\frac{1}{\sqrt{2}}$, which simplifies the equation. Ideally, the points in the low dimensional space are placed such that $q_{j|i}$ is the same value as $p_{j|i}$, meaning with close points x_i and x_j having corresponding points y_i and y_j that are also close. Far points are kept far apart in the low dimensional space. The cost is zero if $p_{j|i} = q_{j|i}$.

Therefore, the dimensionality reduction problem is stated to find the position of each y_i such that $p_{j|i} = q_{j|i}$ for any other point j. SNE defines a cost function and uses gradient descent to find the optimal position for each y_i that minimizes the cost. The cost function is defined as the KL divergence between the probability distributions in high and low dimensional spaces.

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \tag{5.5}$$

The cost is zero if $p_{j|i} = q_{j|i}$ for every pair of points. This means the far (close) points in the high dimensional space is far (close) in the low dimensional space. The cost is large if close points in high dimensional space, $p_{j|i} \approx 1$, are far in the low dimensional space, $q_{j|i} \approx 0$.

The gradient descent optimization updates the position of the points in the low dimensional space using the gradient and momentum terms:

$$\mathscr{Y}^{(t)} = \mathscr{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathscr{Y}} + \alpha(t) \left(\mathscr{Y}^{(t-1)} - \mathscr{Y}^{(t-2)} \right) \tag{5.6}$$

where $\mathscr{Y}^{(t)}$ is the position of the points in the low dimensional space at iteration t, η is the learning rate, and $\alpha(t)$ is the exponential decay factor at iteration t which is a value between 0 and 1. The position of the points in the low dimensional space $\mathscr{Y}^{(0)}$ are initially drawn from a

normal distribution.

t-distribution Stochastic Neighbor Embedding

There are two differences between SNE and t-distribution Stochastic Neighbor Embedding (t-SNE) [4]. First, t-SNE uses a symmetric variant of the SNE similarity functions. For similarity in the high dimensional space

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}, \tag{5.7}$$

where N is the number of points. This is a symmetric similarity measure as $p_{ij} = p_{ji}$ and $p_{ii} = 0$ and $\sum_{i,j} p_{ij} = 1$. A Student t-distribution rather than a Gaussian is used to compute the similarity between two points in the low-dimensional space,

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + ||y_k - y_l||^2)^{-1}}. \tag{5.8}$$

Second, the cost function is defined as the KL divergence between the two distributions of the symmetric similarity measures,

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{5.9}$$

The derivative of C with respect to y_i turns out to be a simple equation,

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (y_i - y_j)(p_{ij} - q_{ij}). \tag{5.10}$$

The partial derivative equation is interpreted as a sum of forces pulling y_i toward y_j or pushing it away, depending on whether j is observed to be a neighbor more or less often than desired.

The algorithm of t-SNE is presented in Algorithm taken from the paper [4], the equation numbers refer to the equations in the original paper. p_{ij} is computed and the positions of the

points in the low dimensional space are drawn from a Gaussian distribution before the iterative gradient descent optimization in the loop. Inside the loop, the positions of the points in the low dimensional space is adjusted to reduce the cost.

Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

Data: data set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$,
cost function parameters: perplexity $Perp$,
optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, ..., y_n\}$.
begin
  compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 1)
  set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
  sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, ..., y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
  for *t=1* **to** T **do**
    compute low-dimensional affinities q_{ij} (using Equation 4)
    compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t) \left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)} \right)$
  end
end

Hinged t-SNE

This section discusses my own modification of t-SNE, specifically designed for musical user interfaces [108]. The name hinged t-SNE for this method is because there are some fixed points, and the remaining points gradually move until the cost function is minimized, with the position of the fixed points as the constraints.

A free variable in the t-SNE optimization setting is the relative position of the clusters in the low dimensional space while constraining the relative distances. For a pair of points that are far in the data space, it is easy to see q_{ij} is close to one. Therefore, as long as the two points in the low dimensional space stay far from each other relative to the other distances, p_{ij} will be also close to one, irrespective of the relative position of the two points. If the two distant points represent the centers of two clusters, it follows that the cost function can be minimized regardless of the relative position of the clusters. In this section, I propose a method to directly control the

relative position of the clusters. In the following experiments, I demonstrate the effectiveness of the method.

In the modified t-SNE optimization method, a single control point is selected from each cluster with a desired position. The control points, using linear interpolation, are moved from their initial position to their destination within half of the optimization steps in the t-SNE algorithm. More specifically, the update equation for the control points is

$$y_m^{(t)} = y_m^{(T)}(2t/T), \tag{5.11}$$

where $y_m^{(t)}$ is the position of the control point m at time step t, and T is the total number of optimization steps. The positions of the control points are not updated by the t-SNE optimization algorithm within the first half of the optimization steps, but the other points are. In the second half of the optimization steps, all points, including the control points, are updated by the t-SNE optimization method.

In the following experiments, 100 points are placed at each vertex of a regular tetrahedron with an edge length of 8. Then, using t-SNE and the proposed method, the points are mapped onto a two-dimensional space. Independent random noise drawn from a normal distribution with a variance of 1 is added to the 400 points. The perplexity parameter of t-SNE is set to 30.

In the first experiment, the vanilla t-SNE algorithm is used as the baseline. Fig. 5.2 shows the clusters in 2d and the cost value that is minimized by t-SNE. The optimization steps were intentionally set to a large number, 5,000, to verify that convergence had been achieved. It is evident from Fig. 5.2 that, at about step 500, the cost, 0.31, is close to its minimal value, 0.29.

In the second experiment, using the proposed method, the four clusters are placed at the vertices of a square with the edge length of 100 in 2d. The control points after optimization step 300 are controlled by the t-SNE optimization method. Fig. 5.3 shows the layout of the four clusters in 2d. Note that t-SNE brings the clusters closer to each other and the final edge length of the square is about 40. Even though the desired relative positions are preserved. This suggests

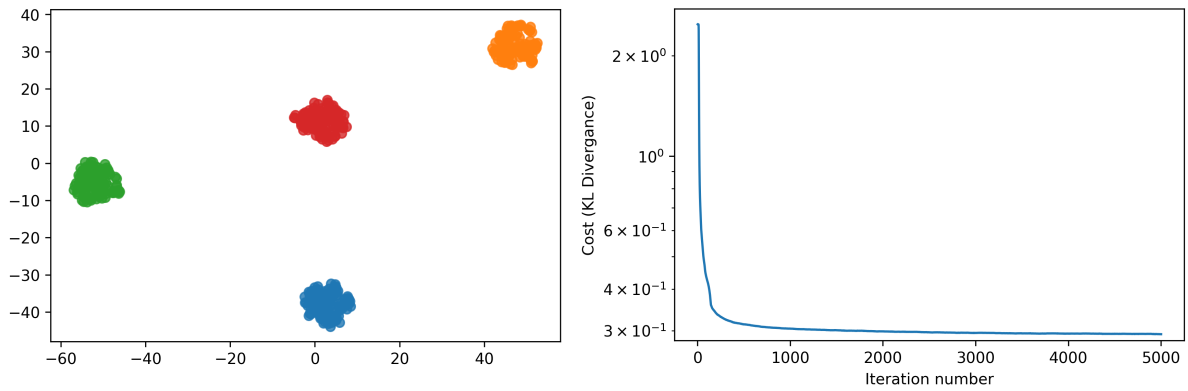


Figure 5.2. Left, the 400 points mapped onto 2d using vanilla t-SNE. The colors represent the vertex of the tetrahedron each point belongs to. Right: the KL divergence cost over 5,000 optimization steps.

t-SNE can adjust the scale, and a precise scale value is not necessary when designing the layout of the clusters with my method. This particular layout for the clusters was chosen to have a non-trivial transformation, i.e., translation, rotation, and scaling, between the layouts in Fig. 5.2 and Fig. 5.3.

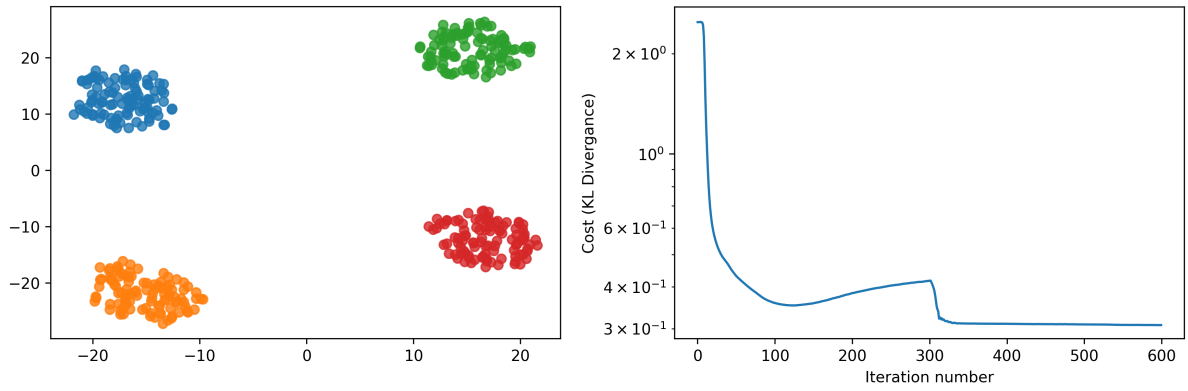


Figure 5.3. Left, the 400 points mapped onto 2d using my proposed method. The colors represent the vertex of the tetrahedron each point belongs to. The four clusters are placed at the four vertices of a square in 2d using my method. Right: the KL divergence cost over 600 optimization steps. At step 300, the t-SNE optimization method takes over the control of the control points.

The loss value is plotted in Fig. 5.3 (right). At step 300, the loss value drops within 10 steps to its minimal value. Fig. 5.4 depicts the reason behind this behavior. At step 300, the clusters do not follow the control points and the control points are separated from their clusters.

The reason is that the pull force between the clusters is strong enough to counteract the pull force from the control points. The right subplot in Fig. 5.4 explains this further. By adding an offset to the position of the orange control point of the cluster at the bottom left, the loss landscape can be visualized to find the position for the control point that minimized the loss. The optimal offset is near [25, 25]. Shifting the orange control point by this offset would bring it to its corresponding cluster. At that point, the total cost is only 13% higher than its minimal value due to the other three control points. Therefore, the t-SNE optimization method needs a small number of steps, about 10 steps in this case, to bring the control points to their clusters and drop the cost as shown in Fig. 5.3 (right).

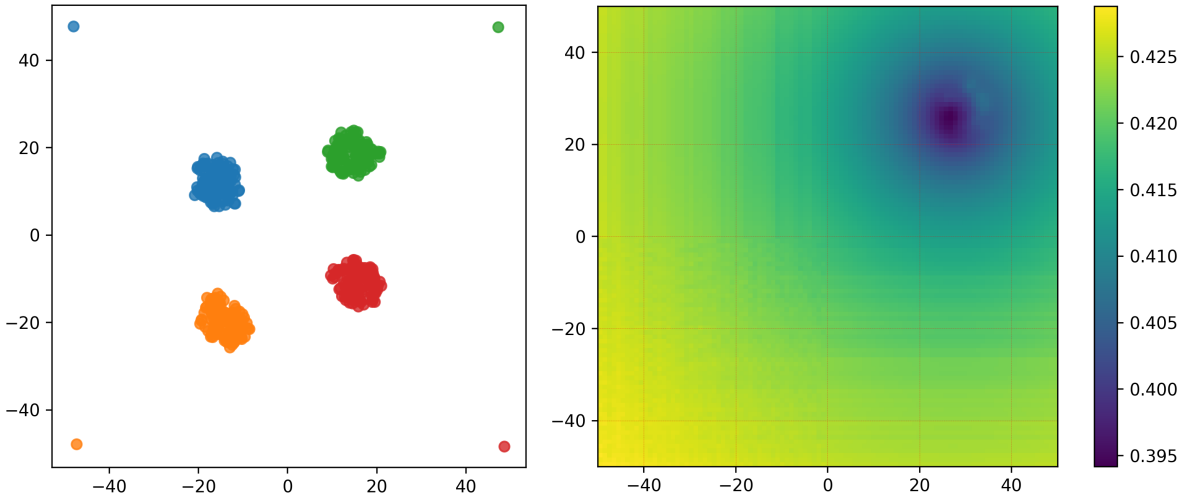


Figure 5.4. Left, the points in 2d mapped using my method at the optimization step 300. Right: the KL divergence cost function value computed for the offset positions of a control point, the orange point at the bottom left in the left subplot.

In the previous experiments, the clusters are well separated from each other. In the third experiment, the variance of the normal noise (which is added to the points in 3d) is increased to two. The fuzzy boundary between the clusters is frequently observed in real applications. Fig. 5.5 shows the layout of the clusters and the cost value over time by t-SNE. The loss value within the first 100 steps is just over 10% larger than the loss value at step 5,000. The fairly rapid drop in the loss value shows t-SNE does not struggle with fuzzy boundaries between clusters.

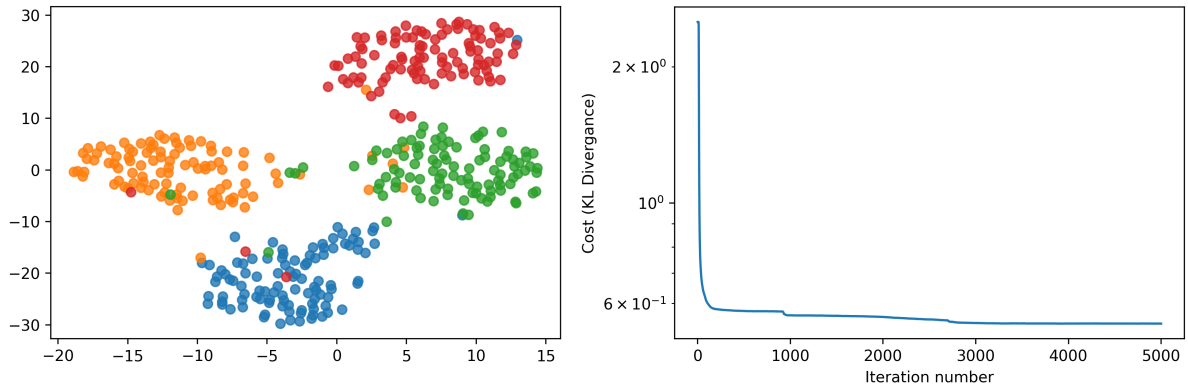


Figure 5.5. Left, the projected points onto 2d by vanilla t-SNE. The boundary between the cluster is observably fuzzy. Right, the cost value at step 100 is about 10% larger than the minimum value at step 5,000.

The same points as in the previous experiment are mapped onto 2d by my method in Fig. 5.6. The method successfully enforces the desired layout of clusters. The plot of cost value against step time shows the optimization manages to converge.

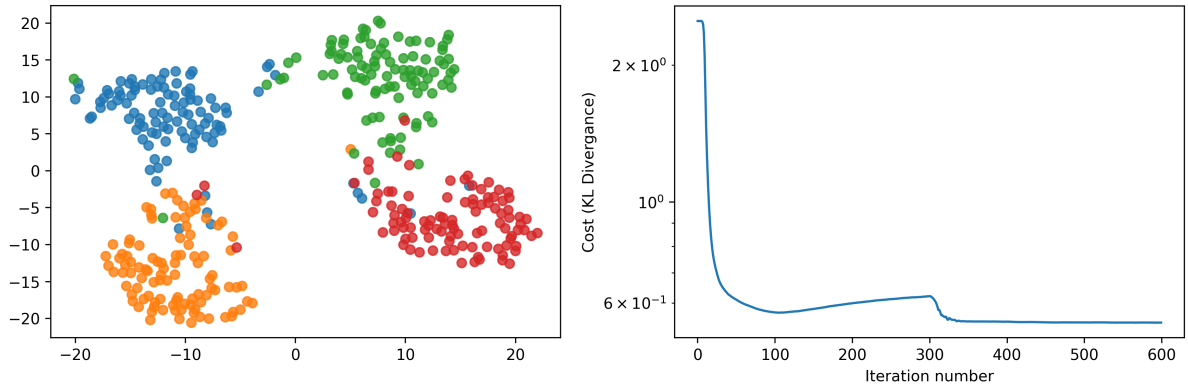


Figure 5.6. Left, the mapped clusters by my method. Despite the fuzzy boundaries, the clusters have the desired layout. Note the non-trivial transformation between this layout and the layout in Fig. 5.5. Right, The cost value is minimized by t-SNE after step 300, and remains stable, suggesting the optimization has converged.

The results of the experiments provide evidence that the proposed method can solve the t-SNE optimization problem with the new soft constraint, namely the layout of the clusters in the lower dimensional space. Controlling the relative positions of the clusters allows a digital musical interface designer to design the high-level layout of the points in the low dimensional

space. This feature can directly be used to increase the usability of the interface by placing the point clusters at relative positions that are intuitive to the user. Later in this chapter, I discuss the application of my modified t-SNE method in a digital drum machine with a set of found sounds.

Uniform Manifold Approximation and Projection

Uniform Manifold Approximation and Projection (UMAP) [109] is a manifold theory based and very similar to the t-SNE algorithm as it also uses a student t-distribution, which is similar to a normal distribution with a lower probability of the median and a heavier tail. UMAP is ideal for machine learning applications because it works perfectly for mapping a very high-dimensional space into a lower one. Other dimensionality reduction algorithms, such as t-SNE or PCA, work well for mapping high-dimensional data to a lower-dimensional space, but not for mapping a very high-dimensional space into a low-dimensional space.

UMAP is also considered a faster algorithm than t-SNE and better for preserving the global structure. t-SNE adjusts each point in the low dimensional space to get a similar distribution as in the original space. UMAP works on trying to find the manifold where the data points lie and constructs a similar manifold with reduced dimensionality.

Neural networks for dimensionality reduction

Neural networks can be trained to map points to a lower dimension. Auto-encoders are unsupervised methods for dimensionality reduction where the encoder transforms the high dimensional data space into a lower dimensional latent space. The reconstruction of the original data from the lower dimensional latent data space to the high dimension input space is done by the decoder. The goal is to minimize the reconstruction error in order to have reconstructed data that is as similar as possible to the original data. Stochastic gradient descent is mostly used to adjust the neural network weights, thereby reducing the reconstruction error. It is possible to train an autoencoder to behave similarly to linear dimensionality reduction methods, such as PCA, by having a single layer of neurons in the encoder without any non-linear activation

function.

Autoencoders as stated, first reduce the dimensionality with the encoder subnetwork E and then reconstruct the input using the decoder part D,

$$D_\Theta(E_\phi(x)) \approx x. \tag{5.12}$$

Therefore, after training an autoencoder, the encoder subnetwork can be used to reduce the dimensionality of the input data.

In autoencoders, the weights of the decoder D and the encoder E are optimized to minimize the distance between every reconstructed $D(E(x))$ point and the input data point x,

$$\min_{\Theta\phi} \sum_{i=1}^{n} ||D_\Theta(E_\phi(x)) - x_i||^2. \tag{5.13}$$

Θ and ϕ are the weights of the decoder and encoder, respectively.

The main problem with autoencoders is that the probability distribution of the training points in the latent space is not estimated. For that reason, it is not possible to sample the latent probability distribution to generate new points using the decoder. Picking a random point in the latent space is likely to generate a noisy output. Variational autoencoders (VAEs) intend to solve this deficiency of the autoencoder and give a better control over the latent space. VAEs also have a compressed latent space, with the latent vectors being drawn from a standard normal distribution. This predefined distribution over the points in the latent space makes sampling from the latent space more suitable for generative models.

Parametric t-SNE

Parametric t-SNE [110] is an unsupervised deep forward neural network for dimensionality reduction. The parametric t-SNE method addresses the problem that with t-SNE, it is not possible to get the embeddings of new data points without having to re-run the t-SNE algorithm on the whole set of points including the new ones. The reason for this limitation is that t-SNE is

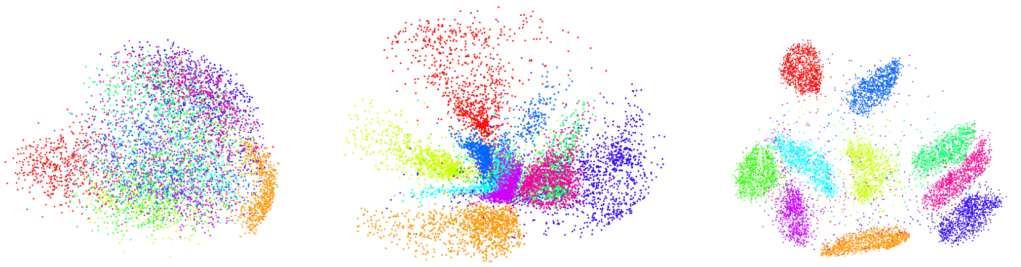
an offline algorithm, that is all input data should be available and processed in a single batch by the algorithm. And t-SNE does not provide a parametric mapping between the high-dimensional data space points and the lower ones, meaning it cannot map any point that is not in the training set using interpolation.

Parametric t-SNE is a t-SNE variant that uses a feed-forward neural network to provide a parameterized nonlinear mapping between the high dimensional and the low dimensional spaces. When comparing parametric t-SNE to autoencoders for dimensionality reduction, it can be observed that parametric t-SNE consists solely of the encoder part, whereas an autoencoder includes both an encoder and a decoder. This difference results in slower computations during training for the autoencoder due to the larger neural network. Also, parametric t-SNE, similar to t-SNE attempts to preserve the distance between the data points in the high dimensional space and in the low dimensional space, mapping the natural clusters in the high-dimensional space to the lower dimensional space by minimizing the Kullback-Leibler divergence. Whereas, the aim of autoencoder is to reduce the reconstruction error between the input and the output data space by the maximization of the variance (difference) in the latent space, which does not preserve the distance between the data points in the original space and the lower space. As a result, vanilla autoencoders have random structure in the lower dimension.

Finally, when comparing parametric t-SNE to PCA, it becomes evident that the linearity of PCA makes it incapable of capturing the true embedding of high-dimensional space, as discussed in the PCA section above. PCA is good at keeping points that are far in high dimensional space far in the low dimensional space, but close points in distance in the high dimensional space are not mapped to close points in distance in the low dimensional space. Figure 5.7 shows the difference in mapping between the PCA, autoencoder, and parametric t-SNE.

Variational Auto-encoders

Variational Auto-encoders (VAEs) [111] use a similar technique as the auto-encoders but with easier control and understanding over the latent space, also known as the lower dimensional



(a) Visualization by PCA. (b) Visualization by an autoencoder. (c) Visualization by parametric t-SNE.

Figure 5.7. A dataset of 28 by 28 pixel images of handwritten digits is mapped to a two dimensional space, using (a) PCA, (b) an autoencoder, (c) parametric t-SNE. It is apparent that parametric t-SNE manages to keep the points belonging to each digit cluster separate from the others. PCA fails to keep the clusters separated. The autoencoder does not achieve a well separated structure in the lower dimensional space as parameterized t-SNE. Source: Reproduced from [110], licensed under CC BY 4.0.

space. For instance, CompressionVAE [112] is an open source library for dimensionality reduction using VAEs with the optional addition of Inverse Autoregressive Flow (IAF) layers that allow the learned distribution to be more expressive, bringing the distribution of the latent vectors closer to the prior distribution. Figure 5.8 shows the effect of IAF on the latent space of VAEs in a sample dataset.

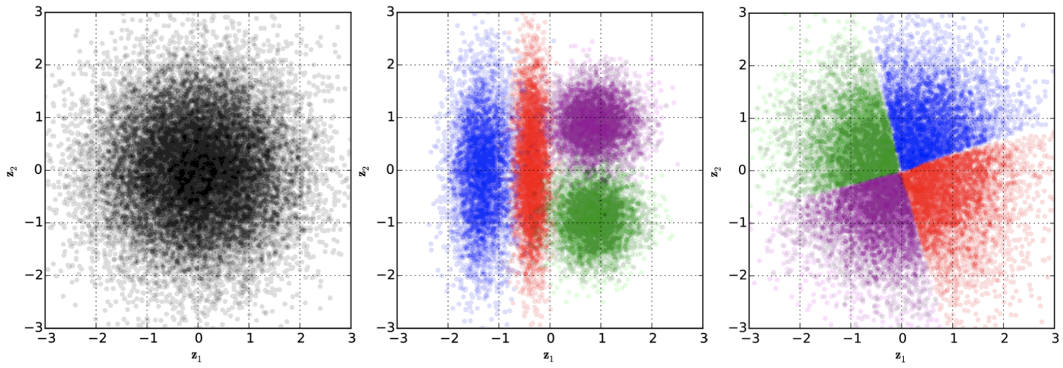


Figure 5.8. Inverse Autoregressive Flow changes the distribution of the points in the latent space to resemble the prior distribution more closely. Source: Reproduced from [4], licensed under CC BY 4.0.

G is the decoder that takes a point in the latent k-dimensional space and generates a point

in the d-dimensional data space.

$$\begin{aligned} G_\Theta : \mathbb{R}^k \to \mathbb{R}^d, \;\; k < d \\ z \mapsto x \end{aligned} \tag{5.14}$$

To help with the derivation, a normal distribution is centered at the point generated by the decoder G, with a fixed covariance matrix.

$$p_\Theta(x|z) = \mathcal{N}(x; G_\Theta(z), \eta I) \tag{5.15}$$

This normal distribution can then be used to calculate the likelihood of the corresponding point x is drawn from it. Then the goal is to maximize this likelihood by optimizing the weights of the decoder Θ. It is also assumed that the encoder, with weights ϕ, maps a point in the data space to a distribution in the latent space. This distribution is not a global distribution, q denotes this distribution,

$$q_\phi(z|x) = \mathcal{N}(z; \mu(x), \sigma(x)I). \tag{5.16}$$

In practice, to simplify the computation, it is a normal distribution with the mean and diagonal elements of the covariance matrix generated by the encoder. In generative models, one can pick a point in the latent space without having an encoder. However, in reconstructive models, the encoder is necessary for generating the z point in the latent space that will be used by the decoder for the reconstruction process.

While VAEs do not explicitly attempt to preserve pairwise distances in the original space and latent space, the latent space still captures some semantic meaning. One method to discover the semantics in the lower dimensional latent space using VAEs is by exploring the directionality in the latent space. For example, when a latent vector is moved in a certain direction, interpretable changes in the reconstructed outputs are observed.

To experiment with VAEs, I trained a VAE on Bach's chorales. The VAE was trained to

reconstruct four measure long piano-rolls. In the first experiment, I aimed to determine whether there are specific directions in the latent space such that, when a latent vector is moved in that direction, certain musical aspects of the piano rolls change. For this experiment, I selected piano rolls that have at least 16 notes with pitches in the last (highest) 10 rows. The latent vectors of those high-pitch piano rolls were generated by the encoder. The average vector of those latent vectors was then calculated. Next, two piano rolls that do not have high-pitch notes were given to the encoder to produce the corresponding latent vectors. The two latent vectors were separately shifted toward the average vector calculated in the previous step. These shifted latent vectors were then passed to the decoder to generate piano rolls. The expectation was that this operation would produce piano rolls similar to the originals, but with some high-pitch notes added.

One question was how far the mean vector should move in the latent space toward the average of the high-pitch piano rolls in the latent space. In the experiments, I moved the latent vector about 1/10 to 1/5 of the distance between the original latent vector and the average vector of the high-pitch piano rolls. For some latent vectors, a shorter distance (1/10 of the distance) produced a better piano roll, similar to the original with some new high-pitch notes. For other latent vectors, a larger distance (1/5 of the distance) produced better results. My hypothesis is that the covariance matrices of the latent vectors are not the same (some are larger than the others). Hence, the distance the latent vector must be moved to observe the same change in the piano roll varies depending on the vector's position in the latent space.

5.4 Dimensionality reduction algorithms in musical interfaces

Digital musical instruments (DMIs) offer controls in an interface that are mapped to parameters of the synthesizer. In some DMIs, there may be more control signals than the number of synthesizer parameters. For instance, in [5], 25 points on the body of a performer are tracked in the three dimensional space, resulting in 75 control values. The 75 values are mapped to 16

values using a VAE. The 16 values in the latent space of the VAE are one-to-one mapped to the parameter of the synthesizer. The proposed system in [5] is depicted in Figure 5.9.

In some DMIs, unlike the above instrument, the number of synthesizer parameters is larger than the number of input control values. The following two DMI discussed fall in this category. It seems that for the DMIs in this category, it is common to use t-SNE, based on the papers I have read.

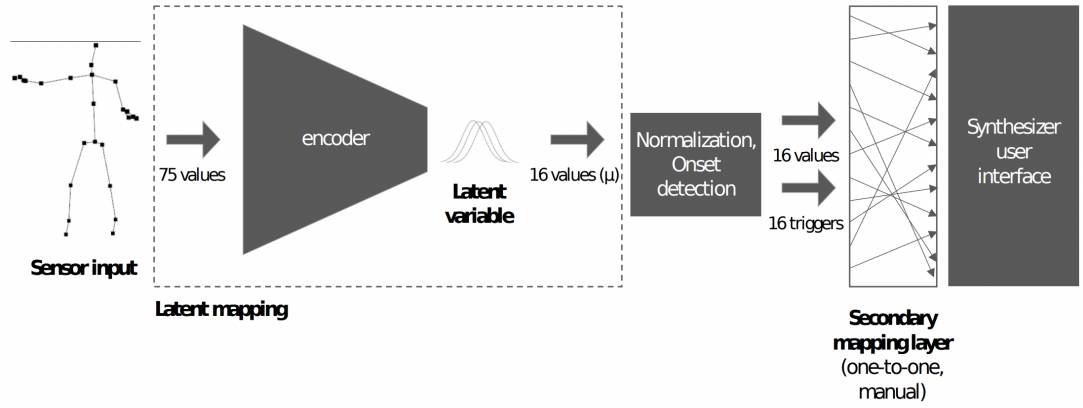


Figure 5.9. The proposed approach in [5] using a VAE to map the control values from a 75-dimensional space to a 16-dimensional space. Source: Reproduced from [5], licensed under CC BY 4.0.

5.4.1 MusicMappr musical interface

The authors in [113] propose a new interface, named MusicMappr, for remixing music. MusicMappr splits a given audio file into 300 millisecond chunks, and computes the mel-spectrograms of the chunks. t-SNE is used to visualize the spectrograms on the screen as two dimensional points. The user can click on a point and the corresponding sound is played. One downside of t-SNE in this application is that the point clusters do not have a predefined layout on the screen. For example, one time the high pitched sounds may end up at the top left corner, another time the low pitched sounds may end up at the top left corner. The user needs to spend time to discover the layout of the points before remixing music. By setting the position of some of the spectrogram points on the 2d screen using Hinged t-SNE, for example, the user would

know the point corresponding to the first 300 ms windows is at the top left corner, and the similar sounding excerpts are always at the top left corner.

5.4.2 the Infinite drum machine musical interface

The Infinite Drum Machine [114] is a novel interface for a drum machine using found sounds, which allows the user to set the rhythm and sound for the sequencer and select points on the screen corresponding to the short sound files. The authors provide about 4,877 audio files with a sampling rate of 44,100 Hz, and each WAV file is about a quarter of a second long. Using short-time Fourier transform (STFT) from the Python library Librosa, the spectrograms (magnitudes) are extracted for each audio file. Every 10 frequency bins are averaged then the spectrogram is normalized to have an amplitude between zero and one. Finally, the first 32 columns (time) and 32 rows (frequency) are taken as the fingerprints of the audio files. In Figure 5.11 the fingerprints of 16 randomly selected audio files are presented.

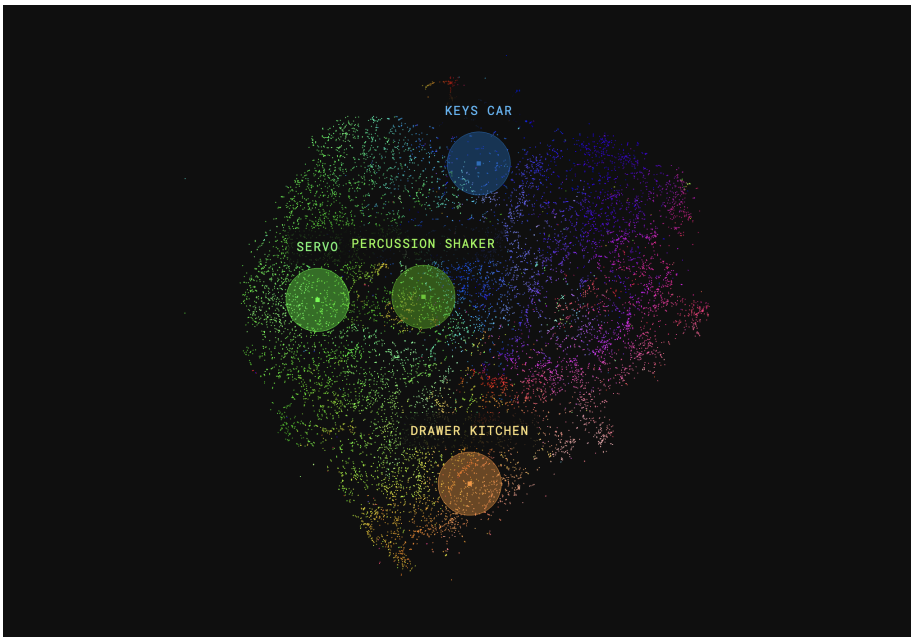


Figure 5.10. The interface of the infinite drum machine. Each point on the screen corresponds to a sound clip. Selecting four points for the four channels of the drum machine, the sequencer can also be used to modify the rhythm for each channel.

In the interface, there is a shuffle button, and when it is clicked, a random sound is

assigned to each of the four channels of the drum machine. However, the assignment is not completely random because a processing script measures the similarity of each sound to the four drum samples (kick, snare, hi-hat, open) using the computed fingerprints.

The way the similarity of each sound and the four drum sounds is measured in this interface by extracting two features from fingerprints using librosa. Specifically, the two methods `spectral_centroid` and `spectral_bandwidth` are used to extract the centroid and bandwidth of the energy distribution and form a two dimensional vector for each sound. The Euclidean distance between the 2d vectors for each sound and each of the four drum samples is then measured. Then the drum class (kick, snare, hihat, open) with the shortest distance is assigned to each sound. An interesting experiment would be to replace the two features with some other features such as inharmonicity, spectral roll-off, and spectral flux (which measures how noisy the spectrogram is) [115].

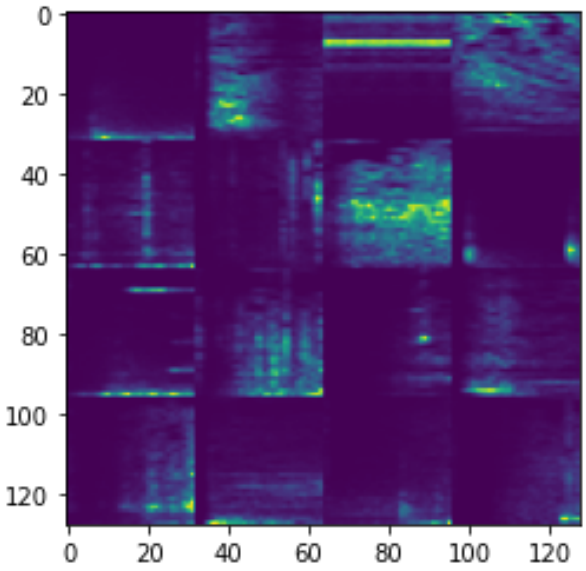


Figure 5.11. The fingerprints of 16 audio files from the infinite drum machine. Each fingerprint is a 32 by 32 matrix. Each matrix coarsely shows where, across time and frequency, most of the energy is distributed. The fingerprints are used to tell which sound files are similar to the four reference sounds for kick, hat, snare, and open.

To determine the location of the points representing the audio files on the screen, t-SNE is used to map the spectrograms to a 2d space. The extent of the points in two dimensions is

then measured and is scaled to fill the screen without changing the aspect ratio of the points' distribution. A sample screenshot is shown in Figure 5.10. The downside of using t-SNE for this interface is that the cluster containing points that sound like kick, for example, does not have a clearly defined position. The cluster of points ends up at a random location. I replaced t-SNE in this interface with my proposed Hinged t-SNE method and defined a preset location for one point from each of the four clusters. This configuration is illustrated in Figure 5.12. The user then can have a general idea of how the sounds are distributed on the screen.

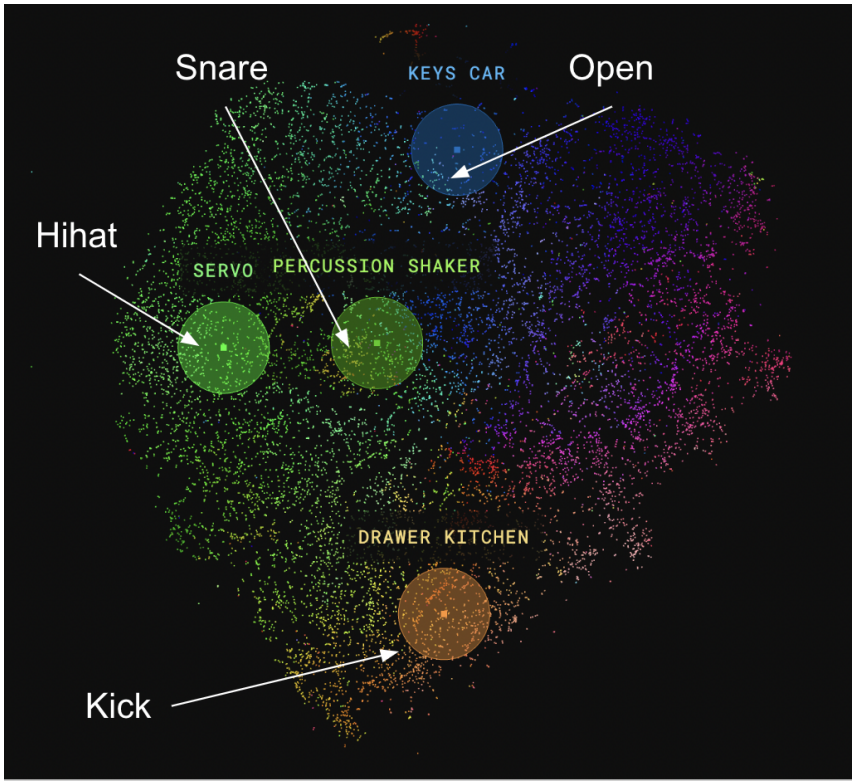


Figure 5.12. One point from each sound cluster is controlled by Hinged t-SNE to have a defined location for each cluster. A short video of interacting with the interface is available at https://youtu.be/rGw3PUAd6a8.

The takeaways from this literature review that guide my own research are twofold. First, there is a large number of dimensionality reduction approaches and methods and almost always they are used in DMI as they are without any modification specific to this use. I think this is a great research opportunity to tweak these methods for use in DMIs to have more intuitive,

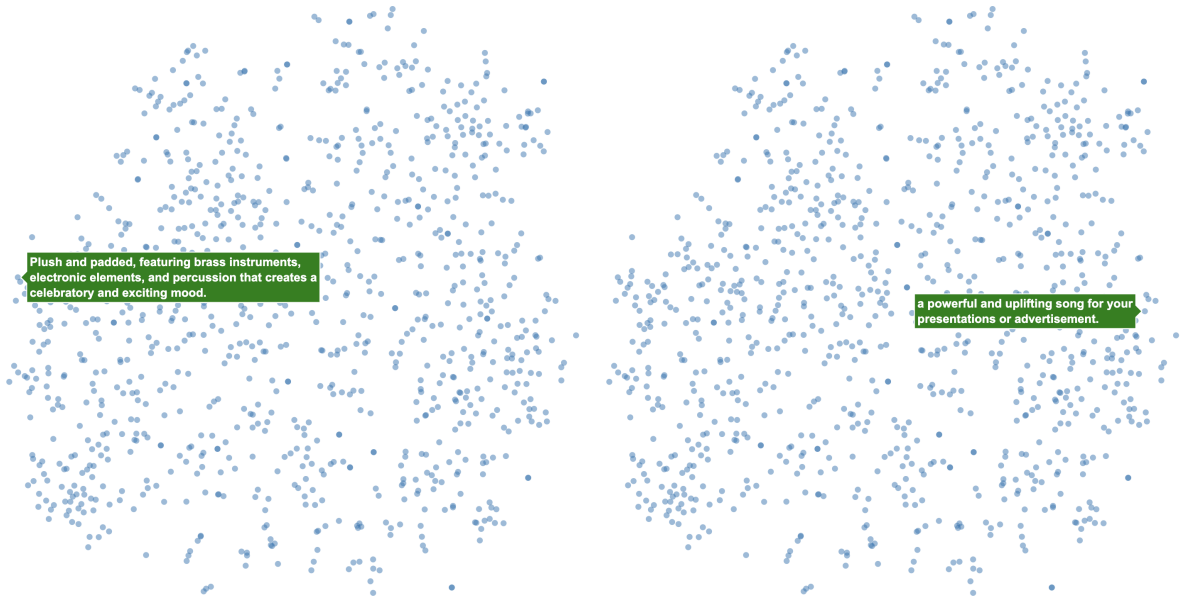


Figure 5.13. Using Hinged-TSNE to arrange the prompts on the screen, with the tempo of the pieces increasing from left to right. The user can explore the prompts to understand, for instance, what level of musical description details is understood by the text-to-music model.

flexible, and controllable musical interfaces. My work on Hinged t-SNE was motivated by this observation. Second, generative models for live performance and offline symbolic music generation could possibly benefit from customized dimensionality reduction methods.

5.4.3 Text-to-music interface

Chapter 4 presented a method to generate long-form music using an LLM and a text-to-music model. It is possible to use the same method for co-composition with a user by adding an interface to modify the text prompt for each section of the music. The weakness of such an interface is that the user is not aware of the prompts that the text-to-music model is trained on. For instance, what level of detail in the musical description is understood by the model. Trying to explore the space of prompts by trial and error can be frustrating and lead to a non-user-friendly interface.

One solution is to present a large number of training prompts on the screen, which can be explored by the user. Using a text encoder, in my experiments I used OpenAI's text-embedding-

3-small (https://platform.openai.com/docs/guides/embeddings), to get the corresponding embedding vectors for the prompts. Using t-SNE, the embedding vectors are reduced to two dimensions and displayed on the screen. The user can interactively explore the points, read the prompts, and find the region with prompts closest to what they intend to use. Using Hinged t-SNE it is possible to create a more intuitive interface, for instance by placing the prompts for pieces with a low tempo on the left, and the pieces with a high tempo on the right. Figure 5.13 presents the interface from my experiment with this approach.

5.5 Conclusions

t-SNE is one of the most commonly used dimensionality reduction methods, applied in a wide range of fields, from exploring biological data to designing musical interfaces. In this chapter, I proposed a new method that adds soft constraints to t-SNE without impacting its original loss function. This extension allows for a more intuitive arrangement of the points by the designer.

Chapter 5, in part, is a reprint of the material as it appears in International Conference on New Interfaces for Musical Expression Proceedings 2022, Atassi, Lilac. The dissertation/thesis author was the primary investigator and author of this paper.

Chapter 6

Allocentric and Egocentric Controllers: Similarities and Differences

6.1 Introduction

This chapter contains my 2022 paper published in Leonardo [116].

Gestural instruments can be divided into two categories based on controller type: that of the egocentric controller, centered on and following a point of the performer's body, and the allocentric controller, using a stationary reference frame. This article discusses (1) the similarities and differences between egocentric and allocentric controllers for gestural instruments from the perspective of performer and instrument designer, and (2) the affordances and constraints of egocentric and allocentric controllers as they, to a large degree, define the characteristics of an instrument. The author presents the initial results of a subjective experiment to encourage future discussion and study of the subject.

Digital musical instruments (DMIs) consist of three main components: gestural controllers, sound engine, and a mapping strategy or function that connects the gestural controller to the sound engine [117]. The gestural component is as important as the other components in specifying the characteristics of an instrument. This is why many aspects of gestural controllers have been the subject of extensive study. For instance, multiple properties of the gestural controller can possibly affect the longevity of an instrument's use [118]. Some evaluation methods have also been proposed [117, 119] to compare controllers. Moreover, novel methods for providing

visual feedback for virtual gestural controllers have been proposed [120]. RGBD cameras, also known as depth cameras, are a common type of sensor that have been used in gesture based digital music interfaces in the past several years. For instance, in 2012, Sentürk et al. designed an interface using Microsoft Kinect to move blocks that control the music [121]. Most commercial depth cameras - either natively or through a third-party software library - are able to detect and track body joints. This body joint tracking has several limitations. The versatility of such a sensor can outweigh its limitations for use in a gestural controller. In this project, a depth camera is used as a sensor that allows us to experiment with the reference frame of the gestural controller. The reference point, or more generally the reference frame, can be attached to the performer's body (i.e., egocentric) or to an external object, referred to as allocentric here. The latter approach is much more commonly used, for instance, see [122, 123, 124]. The term egocentric is chosen in this paper to suggest the connection between this reference frame and the concept of egocentricity in the cognitive science literature [125]. Relatively, the egocentric approach has been explored less. For instance, in one of the instruments designed by Mainsbridge and Beilharz [126], the distance between the arms and the torso controls the sound volume. Consequently, a comparison of the two reference frame types has not gained the attention it deserves in the literature

A gestural controller that allows switching between the two reference frame types makes it possible to directly compare them in practice. Perez et al. [127] divide the approaches to the design of digital musical instruments into instrument-based and composition-based approaches. In the former, the novelty of the instrument interface is of importance to the designer. In the latter, an interface that is best suited to the performance of a certain composition or multiple compositions is of importance. The aim is to make an informed explicit decision on which type of reference frame an instrument should use considering the design requirements for either an instrument-based or a composition-based approach. Therefore, identifying the aspects of the instrument that are affected is necessary in order to compare egocentric and allocentric coordinate systems. Magnusson [128] discusses the importance of affordance and constraints when evaluating a digital musical instrument. Additionally, he emphasizes that these factors

need to be studied from the perspective of the instrument designer, composer, and performer (idiomatic gestures are a related idea [129]). The constraints help to reduce the freedom, as too much freedom and too little freedom for the performer can render the instrument unappealing. Similarly, in this paper, the constraints and affordances of the egocentric and allocentric reference frames in a gestural instrument are compared heuristically. The source code developed during this project is shared on the author's GitHub account, https://github.com/Lilac-code/ALLOCENTRIC-EGOCENTRIC-CONTROLLERS.

6.1.1 Gestural Controller

A virtual instrument is created using an Azure Kinect DK. The body joints are tracked using the Azure Kinect Body Tracking SDK. The C++ source code for the body tracking code transmits the position of the joints as OSC messages. In a Supercollider script, the joint position data are processed and mapped to the synthesizer parameters. The highest frame rate of an Azure Kinect DK is 30 frames per second. This low frame rate can cause audible artifacts if the tracked joints are used to directly control the sound parameters, such as the pitch and volume of a virtual Theremin. Smoothing the 3D position of the joints is an option using the Kinect SDK, but this still has a low frame rate, leading to audible artifacts when controlling the pitch using quick hand movements. For that reason, in this implementation the pitch and volume values are smoothed in Supercollider, that completely eliminates the sound artifacts. While this workaround is needed owing to the limitations of the used 3D tracking technology being used, it does not interfere with the experiments with egocentric and allocentric reference frames.

6.1.2 Allocentric coordinates

The allocentric coordinate system could be divided into two types. In the first, the reference frame is shifted in front of the original coordinate system at a fixed distance. This distance from the depth camera is added to allow the performer to be positioned away from the narrow tip of the camera view pyramid where the field of view covers a small volume. In the

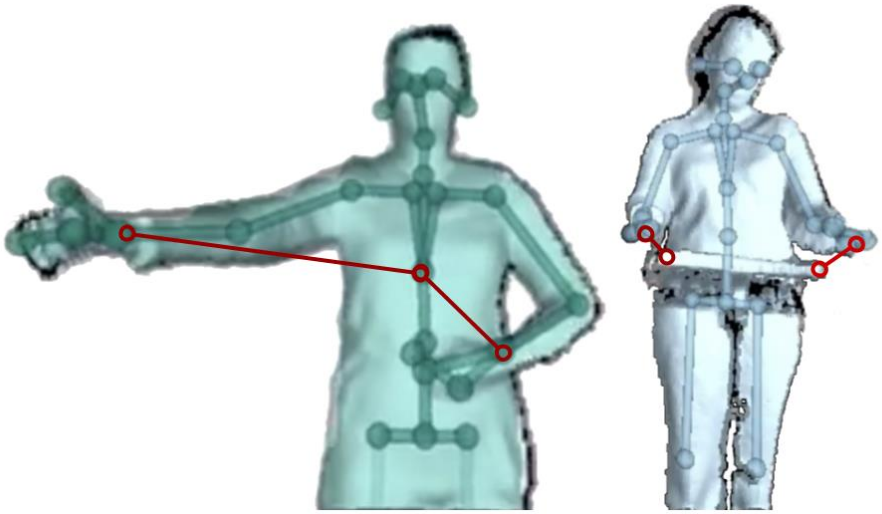


Figure 6.1. Left: The egocentric reference frame is attached to the tracked chest point. The red line is the distance between the wrist to the origin in the 3d space. Right: the allocentric reference frames are attached to points in space, one for each arm.

second type, the allocentric reference frame is attached to an object, similar to the Theremin's antennas. The calibration process starts by pressing the S key on the keyboard. After three seconds, the 3D position of the two hands is sent as an OSC message to Supercollider to record as reference points. Subsequently, the stream process starts, which continuously sends the 3D hand positions. In this case, one can place an object so as to provide visual feedback at the location of the reference points, as illustrated in Figure 6.1 (right).

Implementing a complete transformation of the reference frame with rotation and translation for the allocentric system is challenging. The challenge is estimating the rotation from some visual cues, but as the rotation is relative between the camera reference frame and allocentric frame, one can adjust the relative physical rotation between the camera and object that the allocentric reference will be attached to as part of tuning the system before performance. Therefore, for this paper only translation for transforming the allocentric reference frame is implemented.

As an instrument designer, the remaining problem is how to communicate the location of the two ends of the motion range to the performer. If no visual cue is provided, the performer

needs to rely on the audio feedback to find the two points. Placing a marker on the floor is a commonly used option.

Another problem that needs to be addressed by the designer concerns setting a range for the distance of a tracked joint to the origin. To reduce the noise level, the motion range is scaled down to reduce the noise to a negligible level. This, in effect, reduces the resolution of the joint position estimate. Therefore, to increase the control range, the motion range needs to be increased. For this experiment, the hand motion range is chosen to be 50 cm. The mapping function of the controller can take, as input, the position of the tracked points, the distance between the points, or both.

In both types of allocentric frames, the Euclidean distances can be computed from the tracked points to the origin point in the reference frames. Another option is to compute the distance to a line. This is similar to the Theremin. Given that the camera can be positioned in the desired relative position, the lines can be along a single axis. Next, the point to line distance is determined by computing the Euclidean distance in the 2D space. The distance to plane can also be computed by the distance in one dimension.

6.1.3 Egocentric coordinates

In the egocentric reference frame, the Supercollider script calculates the distance between each hand and the chest points. There is also a flag in the Supercollider script that switches the mode to measure the distance between each hand and the line passing through the chest and navel points. It is also possible to measure distances from a plane in the reference frame.

Transforming the original reference frame to the desired egocentric one is carried out in two steps. One joint (point) is chosen as the origin, and then the distance from all other points to this origin point is calculated. In the case of the distance-based controller, that is all that is needed. For the distance-to-line case, for example, the distance between the hand and the spine, the perpendicular line to the desired line that passes through the point is determined. The dot product of the vector from the origin to the hand point and a unit vector on the perpendicular

line yields the distance between the point and the line. For complete transformation of the point coordinates into a new coordinate system, after choosing the origin joint, two other joints that belong to two orthogonal lines are also selected. These points form two vectors. The orthogonal vector to these two vectors using the external product is computed. This computation is done every time the updated point positions are received via OSC messages. As a result, the three vectors remain attached to the performer's body.

The three vectors form the basis of the new egocentric reference frame. To transform the point positions, first, the vector to each point from the new origin is formed, and then this vector is projected onto the three basis vectors.

Unlike the allocentric controller, a preset distance for the motion range is not used. The arm length is used as the motion range of the hand point. The forearm and upper arm lengths are calculated and added to calculate the full arm length. The measurement noise adds a negligible inaccuracy to this calculation.

In the egocentric frame, it is also possible to calculate both positions and distances. The noise level of positions is amplified compared to the allocentric controller as the noise from multiple tracked points adds up. For distances, the noise level is on par with the allocentric controller, as the noise from the egocentric reference frame cancels out when subtracting two positions.

Calculating the distance between point to a line and a plane in the egocentric system is done similarly to calculating that for the allocentric system, with the difference being the line and plane points, are tracked body points. The more body points used in the equation, the more noise is amplified. Therefore, the point distance to plane has a higher noise level than the point distance to line. Moreover, the point distance to another point has a lower noise level than the point distance to a line.

6.1.4 Effort-based mapping

The importance of effort or energy in digital musical interfaces has been discussed in the literature [122, 130, 131]. It has been argued that particularly for an expressive instrument, the energy should be transferred from the performer to the instrument. This requires that the signals from the sensors are mapped to the sound engine parameters such that the input energy is mapped to the output energy. Effort or energy is not only motion. Remaining in a non-rest state requires energy. For instance, keeping a leg or arm in certain positions requires effort and energy [122]. Therefore, it is also common to map the amount of deviation from the rest state to a sound parameter [130, 131].

Measuring the speed of motion in both reference frames is equally simple. However, measuring the energy in a non-rest state presents several difficulties in the allocentric system, not in the egocentric system. For instance, the rest state of an arm when a performer is standing is parallel to the torso with the hand being close to the hip. The higher the hand is raised, the more effort is required. Thus, the distance between the hand and the hip is almost proportional to the amount of exerted effort.

6.1.5 Affordances and Constraints

The affordances and constraints of an instrument define some of its main characteristics [128]. The affordances shape the way the performer interacts with the instrument. The constraints shape the way the performer performs with the instrument. The instrument controller mostly defines the affordances of the instrument and has a sizable influence on the constraints. Therefore, the following sections compare the affordances and constraints of the egocentric and allocentric controllers.

Neither of the two controllers provides physical interfaces to interact with. In the allocentric system, the presence of an object resembles the interaction between the performer and the object. It is very likely that the performer thinks of an analogy as soon as they start

interacting with the instrument. As their arm gets closer to the object, they expect control over some sound parameter. This might feel natural, and the object feels inviting to the performer to interact with the instrument. Using the egocentric controller, the experience feels very novel. The lack of any external object means there is nothing the performer is drawn to interact with. This is particularly the case during the first few minutes of interacting with the instrument, when the performer likely attempts to imitate the same behavior they exhibit with the allocentric system - by moving the arms in the direction of the camera.

Sonic affordance has been argued to be possibly as strong as object-based affordance; and sound can suggest gesture even when the sound has no cultural association [132]. This possibly explains why after spending some practice time with an egocentric controller, the unfamiliar new interaction turns quickly to a more familiar interaction, despite the lack of object-based affordance. The sonic affordance of egocentric instruments, in my opinion, represents an interesting research venue for further exploration. This possibly, in part, explains why the performer develops an understanding of the instrument over time, as it is not limited to only the causal relationship between action and sound [133].

The constraints that define the characteristics of an instrument are arguably at the sound and mapping engine level [128]. The objective constraints of the interface have less effect on the virtuosity of new musical instruments. The constraints of an instrument also motivate the performers to explore the instrument and find its limits [134]. Vasquez et al. also study the impact of the constraints [135].

The possible set of constraints at the mapping level for egocentric and allocentric interfaces is the same. This is due to the fact that the same measurements are available for both interfaces - that is distances, relative positions, speeds, and accelerations are the measures inputted from both interfaces to the mapping method of the sound engine. Therefore, this section compares the objective constraints of the two interfaces.

One noticeable difference between the two systems is that. with the allocentric controller, the relative distance or speed between the hand and torso does not matter. The performer is free

to move their torso. With the egocentric controller, this freedom is replaced with freedom in body movement as long as the relative position is controlled. Exploring this egocentric constraint as a performer feels less like learning a new acoustic instrument and feels more like a novel experience.

Another noticeable difference in constraints between the two controllers is the number of body joints at the performer's disposal to achieve a certain position. With the allocentric controller, each body joint and the entire body can participate and help the reach. With the egocentric controller, only the shoulder, elbow, and wrist collaborate to reach a certain hand position from the chest.

6.1.6 Discussion

The difference between the egocentric and allocentric controllers is sufficiently great to lead to a large difference in the characteristics of an instrument. In this paper, I shared my experience of a performer interacting with the two controllers. Studying the experience of a larger number of experienced and inexperienced musicians over a longer practice time with both controllers opens up an important research area. Such research can answer interesting research questions such as whether an egocentric gestural instrument is more appealing to performers.

In this study a depth camera is used as the sensor of the controller. This sensor has some inherent limitations. The camera is fixed in a specific location, and the performer has to remain within the camera's field of view. Occlusion of the performer can interfere with the data input of the controller. Additionally, and perhaps most importantly, the camera is not capable of tracking finger movements accurately unless they occur within a very short distance from the camera. Other sensors are also commonly used for gestural instruments, including gloves [136, 137, 138]. Combining multiple sensors has also been shown to have advantages for gestural instruments, for instance finger and arm tracking [139]. Exploring various sensor technologies to build and compare ego-centric and allocentric controllers is an area of research that would shed more light on the performers' experience with such controllers.

This chapter discussed the differences and similarities between the two controllers from the perspective of the instrument designer and the performer. As important, if not more important, is a comparison of the two controllers from the audience's perspective. For instance, future studies should attempt to answer the following questions: Is one or the other of the controllers more appealing to the audience? Does one or the other make it easier to demonstrate virtuosity?

A developmental approach is adopted to compare egocentric and allocentric gestural controllers. At the implementation level, there are differences that are important for designing a usable digital musical instrument. There are also significant differences between the two that are noticeable to the performer. In this work my own findings are presented after practicing with the two controllers while maintaining the same mapping strategy and sound engine for both. Thus, this review is limited. A subjective comparison of the egocentric and allocentric gestural controllers by a larger group of performers remains an interesting research question left for future studies.

Chapter 6, in part, is a reprint of the material as it appears in Leonardo Music Journal, Atassi, Lilac, MIT Press 2022. The dissertation/thesis author was the primary investigator and author of this paper.

Chapter 7

Human Voice Detection For Search and Rescue Operations

7.1 Introduction

This chapter contains the report written for the Small Business Technology Transfer (STTR) project that was a collaboration between me and Miller Puckette at UCSD and a startup called TrueFace in 2022. The report is kept in its original form with minimal edits in this chapter.

The goal of this project is to investigate whether sound is a viable medium for automatic detection of people in a disaster area. Given that this is a very general research question, we make several assumptions suitable for this feasibility study. We assume the hardware related problems are not concerning, and identifying and solving them can be carried out in the next phase of research. In this study, we do not explore guiding the navigation and control system of an autonomous vehicle. We investigate with a brief audio segment if an algorithm can detect whether human voice is present. The ambient noise turns this into a challenging problem because, in some cases, it masks the human voice.

By mixing some ambient noise with clear human voice audio, we generated a dataset to train and test neural networks for the task. We evaluated the performance of multiple neural network architectures from a fully connected network with two layers to a convolutional network with tens of layers. Informal tests showed that the best performing algorithm in our experiments makes the same number or fewer mistakes at detecting human voices than a human listener

would.

7.2 Data

To create our training and test dataset, we gathered a set of human voice samples and a set of ambient noise samples. With this approach, we can select the relative loudness of the two sounds when combining them and generate the dataset more quickly versus collecting the mixed audio data using a microphone.

The human voice samples[1] are taken from a project with the aim of developing an open source voice command recognition model. The publicly available data is under the Creative Commons License. Each sample is a few seconds long and is recorded in a quiet environment without any audible ambient noise.

The ambient sound samples are taken from the urban sound datasets project[2] which sorts a large number of Creative Commons licensed audio files, taken from another source[3], into several categories. The dataset consists of urban environmental sounds such as car honking, air conditioner noise, freeway noise, and nature. We chose three audio files from this dataset containing street noise, each one is about 20 minutes long.

We use a subset of the human voice samples containing 12 people reading some text, and 5 audio files per person. Each recording is about 5 seconds. For each voice sample, an ambient noise sample out of four is chosen randomly. The voice sample is then added to non-overlapping 5 second windows of the ambient noise resulting in about 240 (20 minutes $\times$ 60 / 5 seconds) samples of voice plus noise. This process yields 14,400 (12 $\times$ 5 $\times$ 240) voice combined with noise samples.

When mixing the sound samples, we calculate the root mean square (RMS) value of each sound sample, and scale the amplitudes to obtain 0,-1, and -5 dB ratios of voice to ambient sound

[1] https://commonvoice.mozilla.org/en/datasets
[2] https://urbansounddataset.weebly.com/
[3] https://freesound.org/

amplitudes. The sampling rate of the voice samples is 16 kHz. The ambient sound samples are resampled to have the same sampling rate of 16 kHz. The mixed sound is then windowed into 30-msec (480 sample) segments for audio analysis.

7.3 Performance metric

The ROC curve (false positive rate vs. true positive rate) is used to evaluate the performance of the algorithms. The ROC curve can be used to compare the performance of algorithms at a false positive rate that is suitable for the application. For example, if the cost of false alarms in a search and rescue mission is high, we can compare the accuracy of the models for low false positive rates. As the cost associated with the false alarm rate in this application and in this phase is impossible, the ROC curve is helpful in predicting the accuracy of the model when deployed in a real mission.

7.4 Experiment one: convolutional neural network

In the first experiment, a convolutional neural network, with five convolutional layers and two fully connected layers, was trained on the data. Batch normalization and dropout were added between the layers to regularize the network. The 2-second input audio signal is divided into 30 millisecond (480 sample) windows, and for each window the 1024-bin spectrum is computed using the Fourier transform. The frequency bins are then transformed to the mel scale with 120 bins. Fig.7.1, left panel, shows the relation between frequency (in Hertz) of the mel scale bins. The right panel of Fig.7.1 shows the same relation but with the logarithm of the frequency.

By applying the mel scale, the lower frequencies are sampled more densely than higher frequencies. This is expected to reduce the amount of data while preserving most of the information in the spectrogram. Stacking the spectrums with the mel scale, we get a 66×120 matrix as an input sample to the neural network. Fig.7.2 shows a sample spectrogram of an audio sample containing voice added to ambient noise that is fed to the neural network.

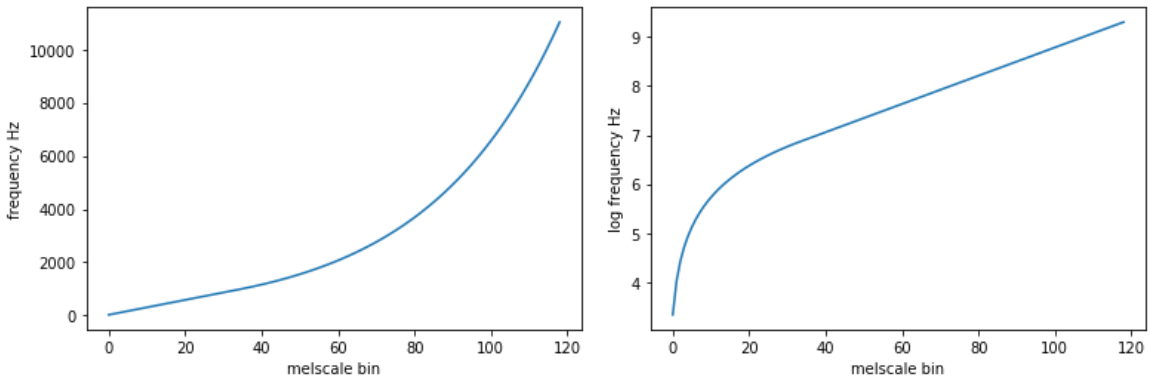


Figure 7.1. Visualizing the relationship between frequency and mel scale bins.

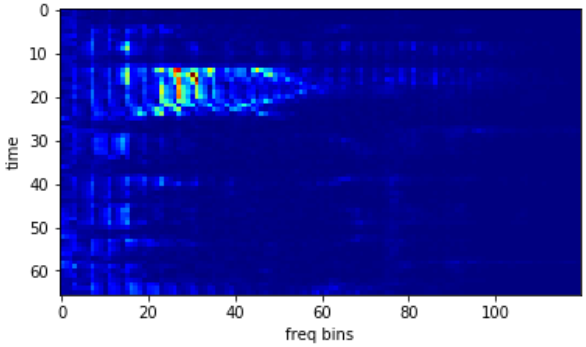


Figure 7.2. Visualizing the spectrogram of an audio clip containing voice and background noise.

The architecture of the network is depicted in the Fig.7.3. The five convolutional layers are followed by Max Pooling layers. The last two layers are fully connected layers, with the last one having two outputs. The cross-entropy loss function and the Adam optimizer are used to train the model for four epochs. The data is divided into two sets, 90% for training and 10% for testing.

The ROC curve of the convolutional network on the test dataset is presented in Fig.7.6. Given the simple architecture, it is remarkable to achieve about 60% true positive rate at zero false positive rate. The weakness of the algorithm is at achieving a true positive rate above 90%, which is only possible at false positive rates greater than 60%.

7.5 Experiment two: ResNet model

In the next experiment, we replace the convolutional network with an 8-layer ResNet model. Deep residual networks [140], or ResNets, were introduced in 2016 with the main idea of encouraging the neural networks to learn the difference between expected output and input. This, in turn, makes training neural networks with a large number of layers easier. ResNet models are known to generalize better than vanilla convolutional networks. The diagram in Fig.7.4 shows the architecture of the network we used in this experiment. The input is the same as in the previous experiment, a 66×120-bin spectrogram. The cross-entropy loss and the Adam optimizer are used to train the network for five epochs.

The ROC curve of the ResNet model on the test dataset in Fig.7.6 shows a noticeable improvement in accuracy compared to the convolutional network above. The main weakness of the model is to achieve the true positive rate of 100% with more than 60% false positive rate.

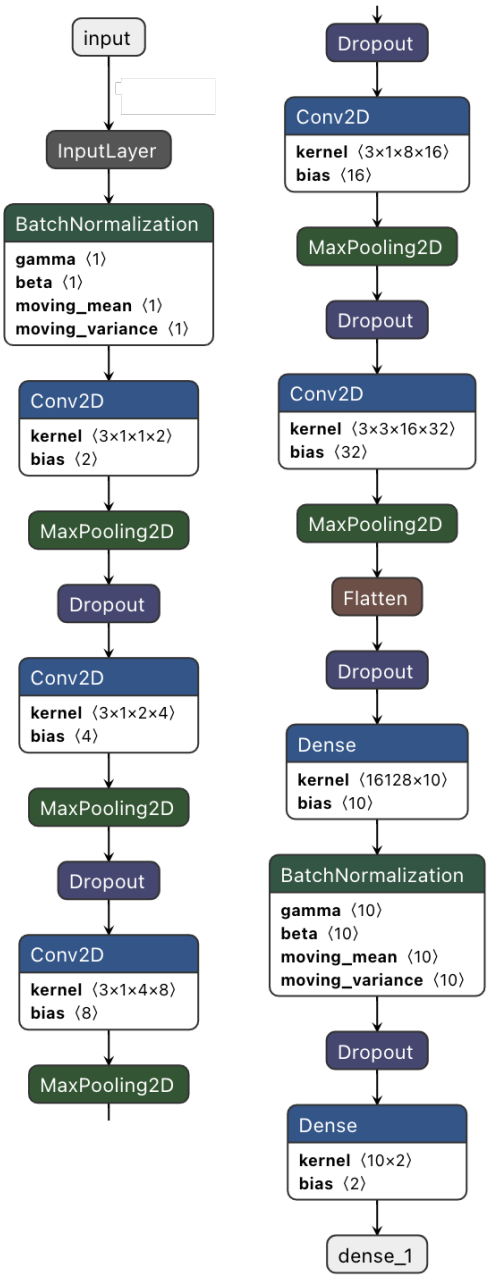


Figure 7.3. The convnet architecture.

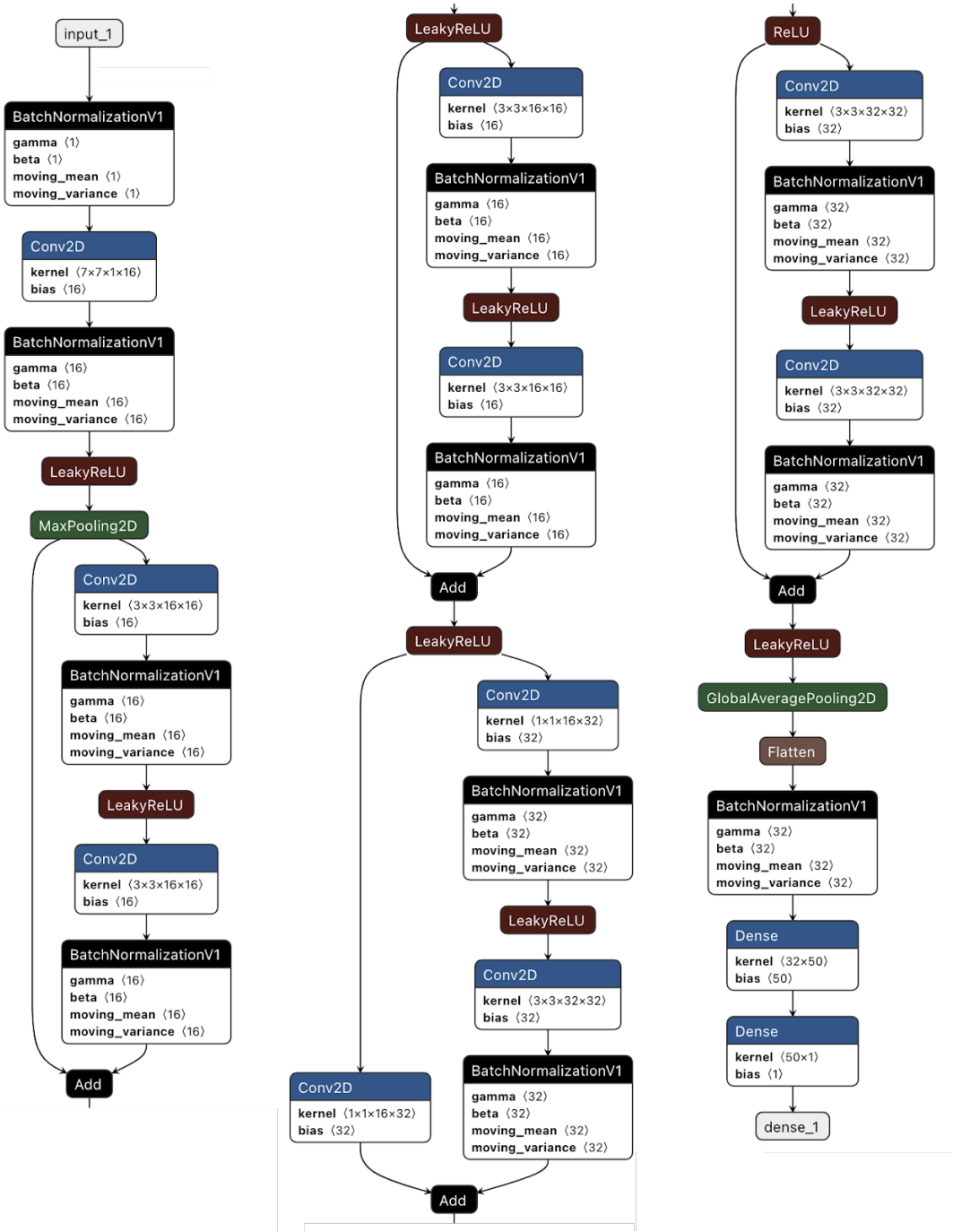


Figure 7.4. The ResNet architecture.

7.6 Experiment three: ResNet model with OpenL3 transfer learning

We used a pre-trained model from OpenL3[4] which is an open source implementation of the L3-Net model[141]. The L3 network is trained to tell whether a given audio file the sound matches a given video frame, that is the audio is taken from the video represented by the frame. A large set of youtube videos are used to train the model.

OpenL3 after computing the fourier transform of the given audio sample transforms the frequency bins to the mel scale. The model has two subnetworks used to extract features from an audio segment and a video frame. The extracted features are then concatenated and fed to a third network that predicts whether the two inputs are a match. Each subnetwork can be used as an off-the-shelf feature extractor to train another model for some other task. This approach is considered a transfer learning method which is often used when there is limited training data for a task. OpenL3[5] provides high-level functions to load and use a pre-trained audio subnetwork which we use in our experiments.

The OpenL3 library provides two sets of models, one set trained on music only videos, and the other set trained on environmental videos. In our experiments, we use a model trained on environmental videos. There are also two sets of models trained based on the embedding size, that is the length of the output vector per audio frame. We use the embedding size of 512 rather than the default size of 6,144. The input audio length is not fixed, and for each 100 milliseconds an embedding vector is generated. In our experiments the audio window length is two seconds. Therefore, for each audio window we have 20 embedding vectors, and the output is a 20 by 512 matrix. These matrices are used to train a resnet model with 9 layers to classify the input as ambient noise with human voice or noise without voice. The diagram in Fig.7.5 shows the architecture of the network we use to process the output of the L3 network and make a decision whether a voice is present or not.

[4] https://github.com/marl/openl3
[5] https://openl3.readthedocs.io/en/latest/tutorial.html

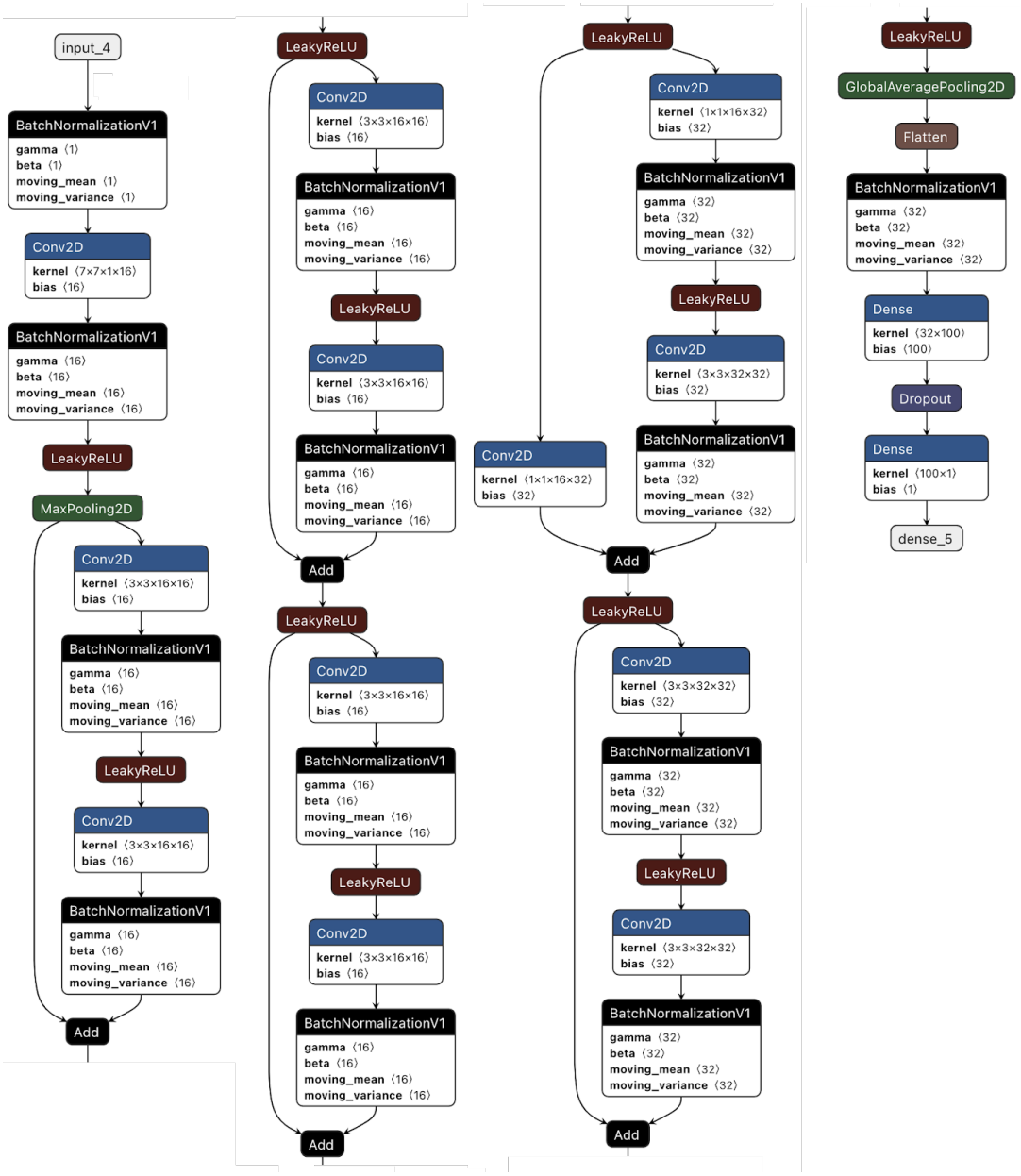


Figure 7.5. The L3 network head architecture.

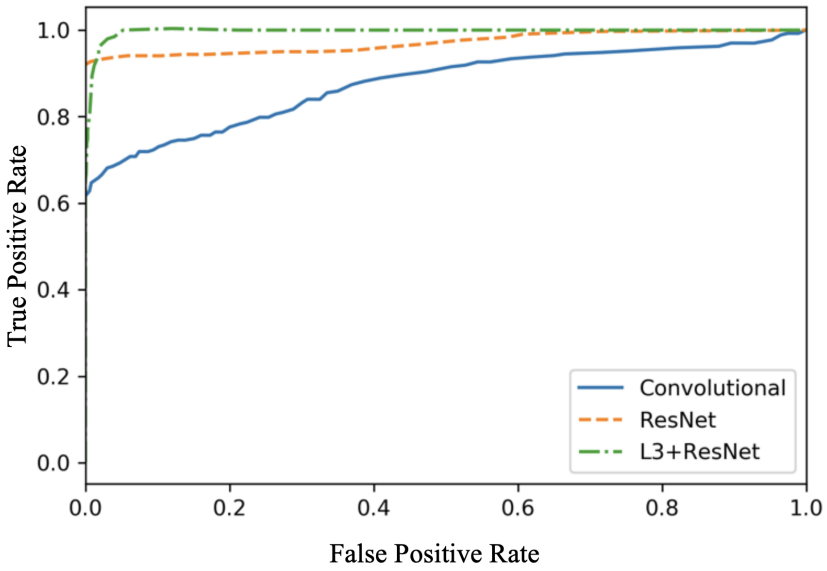


Figure 7.6. The ROC curves for the three experiments. Regular convolutional network performs the worst. The ResNet model performs significantly better. A ResNet on the top of the L3 embeddings has promising performance and outperforms the other two models.

The above plot is the ROC curve on the test dataset. The model can achieve over 65% true positive rate for the false positive rate of zero. If we tolerate a small false positive rate, e.g. less than 5%, we can achieve a high true positive rate, e.g. above 90%. By listening to some of the audio segments that are challenging for the algorithm, it is clear that this is also a challenging task for humans when the voice is masked by loud ambient noise as we cannot hear the human voice in most of them.

7.7 Discussion

Collecting or synthesizing representative data for human voice detection in search and rescue operations appears to be the main challenge. By representative data, we mean audio data that is collected from such missions with people calling for help, for example. Synthesizing or simulating such scenarios and collecting data both lead to low confidence in the generalization of the trained models on real data. Therefore, the better option appears to be using transfer learning to reduce the required training data, and using representative data to evaluate the generalization

of the model.

Should this project be funded for further research to prepare a deployable model, we expect a large amount of effort to be dedicated to collecting and generating data and evaluating the generalization of the models to estimate the confidence in the reported accuracy.

The second challenge we worked on is choosing a machine learning model that yields the expected level of accuracy. Testing a vanilla convolutional neural network provides a starting baseline and an idea of the difficulty of the task. With a ResNet and a handful of residual blocks and layers we observed a large improvement over the vanilla convolutional network. Increasing the number of residual blocks and layers increases the accuracy but at the computation cost at runtime. Given the algorithm is likely to be running on a computer with limited resources, we think further research into the tradeoff between inference speed and accuracy of the model should be carried out in the next phase of the research.

We conclude that: (1) the ResNet model outperforms a convolutional neural network; and (2) system performance is enhanced by using transfer learning. Using ResNet and the OpenL3 transfer learning approach we were able to get results that compare favorably (in informal testing) with human listeners. Designing a formal experiment for evaluating the model's performance against human listeners remains for the next phase of the study.

Chapter 8

Conclusion

This dissertation presents my research on generative music models and musical interfaces. My primary focus has been on reducing the amount of user guidance required by ML-based music generative models, thereby, enabling a wide range of use cases. Despite the progress made, some challenges remain in achieving such a model. One key area of my research has been the development of models capable of generating several minutes of coherent and structured music in various musical forms. My work on generating music with form introduces a new approach that shows promising results.

The research directions I intend to pursue include the development of self-critical music models. Currently, generative models lack the ability to evaluate how appealing and pleasing their music is to a specific audience. However, a generative model that has developed a musical taste aligned with a particular group could critique its own output, deliberate, and improve upon it. This closed feedback loop could have applications beyond self-critical models. For instance, a related research question is whether such models could eventually create entirely new musical genres and forms.

Bibliography

[1] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *International Conference on Machine Learning*. PMLR, 2021, pp. 8162–8171.

[2] C. J. Tralie and B. McFee, "Enhanced hierarchical music structure annotations via feature level similarity fusion," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 201–205.

[3] T. Magnusson, "Designing constraints: Composing and performing with digital musical systems," *Computer Music Journal*, vol. 34, no. 4, pp. 62–73, 2010.

[4] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[5] T. Murray-Browne and P. Tigas, "Latent mappings: Generating open-ended expressive mappings using variational autoencoders," in *NIME 2021*. PubPub, 2021.

[6] A. L. Samuel, "Some studies in machine learning using the game of checkers. ii—recent progress," *IBM Journal of research and development*, vol. 11, no. 6, pp. 601–617, 1967.

[7] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts, "GANSynth: Adversarial neural audio synthesis," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=H1xQVn09FX

[8] S. Bengio and Y. Bengio, "Taking on the curse of dimensionality in joint distributions using neural networks," *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 550–557, 2000.

[9] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ser. ICML'12. Madison, WI, USA: Omnipress, 2012, p. 1881–1888.

[10] C.-Z. A. Huang, T. Cooijmans, A. Roberts, A. Courville, and D. Eck, "Counterpoint by

convolution," in *International Society for Music Information Retrieval (ISMIR)*, 2017.

[11] S. Oore, I. Simon, S. Dieleman, D. Eck, and K. Simonyan, "This time with feeling: Learning expressive musical performance," *Neural Comput. Appl.*, vol. 32, no. 4, p. 955–967, feb 2020. [Online]. Available: https://doi.org/10.1007/s00521-018-3758-9

[12] C. Donahue, H. H. Mao, and J. McAuley, "The nes music database: A multi-instrumental dataset with expressive performance attributes," in *International Society for Music Information Retrieval Conference*, 2018.

[13] B. Li, X. Liu, K. Dinesh, Z. Duan, and G. Sharma, "Creating a multitrack classical music performance dataset for multimodal music analysis: Challenges, insights, and applications," *IEEE Transactions on Multimedia*, vol. 21, no. 2, pp. 522–535, 2019.

[14] Q. Xi, R. Bittner, J. Pauwels, X. Ye, and J. Bello, "Guitarset: A dataset for guitar transcription," in *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018*, ser. Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, E. Gomez, X. Hu, E. Humphrey, and E. Benetos, Eds. International Society for Music Information Retrieval, 2018, pp. 453–460.

[15] Q. Kong, B. Li, J. Chen, and Y. Wang, "Giantmidi-piano: A large-scale midi dataset for classical piano music," *Transactions of the International Society for Music Information Retrieval*, vol. V, 2020, cite arxiv:2010.07061Comment: 9 pages. [Online]. Available: http://arxiv.org/abs/2010.07061

[16] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, "Enabling factorized piano music modeling and generation with the MAESTRO dataset," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=r1lYRjC9F7

[17] J. Gillick, A. Roberts, J. Engel, D. Eck, and D. Bamman, "Learning to groove with inverse sequence transformations," in *International Conference on Machine Learning (ICML)*, 2019.

[18] L. Callender, C. Hawthorne, and J. Engel, "Improving perceptual quality of drum transcription with the expanded groove midi dataset," 2020.

[19] Z. Wang*, K. Chen*, J. Jiang, Y. Zhang, M. Xu, S. Dai, G. Bin, and G. Xia, "Pop909: A pop-song dataset for music arrangement generation," in *Proceedings of 21st International Conference on Music Information Retrieval, ISMIR*, 2020.

[20] R. Louie, A. Coenen, C.-Z. A. Huang, M. Terry, and C. J. Cai, "Novice-ai music co-creation via ai-steering tools for deep generative models," *Proceedings of the 2020 CHI*

Conference on Human Factors in Computing Systems, 2020.

[21] S. Dai, Z. Jin, C. Gomes, and R. B. Dannenberg, "Controllable deep melody generation via hierarchical music structure representation," in *ISMIR*, 2021.

[22] G. Brunner, Y. Wang, R. Wattenhofer, and S. Zhao, "Symbolic music genre transfer with cyclegan," in *2018 ieee 30th international conference on tools with artificial intelligence (ictai)*. IEEE, 2018, pp. 786–793.

[23] O. Cífka, U. Simsekli, and G. Richard, "Supervised Symbolic Music Style Translation Using Synthetic Data," in *Proceedings of the 20th International Society for Music Information Retrieval Conference*. Delft, The Netherlands: ISMIR, Nov. 2019, pp. 588–595.

[24] S. Dubnov, G. Assayag, and V. Gokul, "Creative improvised interaction with generative musical systems," in *2022 IEEE 5th International Conference on Multimedia Information Processing and Retrieval (MIPR)*. IEEE, 2022, pp. 121–126.

[25] J. Nistal, C. Aouameur, I. Velarde, and S. Lattner, "Drumgan vst: A plugin for drum sound analysis/synthesis with autoencoding generative adversarial networks," in *Machine Learning for Audio Synthesis (MLAS) workshop at ICML 2022*, 2022.

[26] E. Deruty, M. Grachten, S. Lattner, J. Nistal, and C. Aouameur, "On the development and practice of ai technology for contemporary popular music production," *Transactions of the International Society for Music Information Retrieval*, vol. 5, no. 1, 2022.

[27] J. Engel, C. Resnick, A. Roberts, S. Dieleman, M. Norouzi, D. Eck, and K. Simonyan, "Neural audio synthesis of musical notes with wavenet autoencoders," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 1068–1077.

[28] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[30] L. Yao, S. Ozair, K. Cho, and Y. Bengio, "On the equivalence between deep nade and generative stochastic networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2014, pp. 322–336.

[31] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in

Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, 2018.

[32] Y. Wu, E. Manilow, Y. Deng, R. Swavely, K. Kastner, T. Cooijmans, A. C. Courville, C.-Z. A. Huang, and J. Engel, "Midi-ddsp: Detailed control of musical performance via hierarchical modeling," vol. abs/2112.09312, 2021.

[33] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A hierarchical latent vector model for learning long-term structure in music," in *International conference on machine learning*. PMLR, 2018, pp. 4364–4373.

[34] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, I. Simon, C. Hawthorne, N. M. Shazeer, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer: Generating music with long-term structure," in *ICLR*, 2019.

[35] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts, "GANSynth: Adversarial neural audio synthesis," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=H1xQVn09FX

[36] G. Mittal, J. Engel, C. Hawthorne, and I. Simon, "Symbolic music generation with diffusion models," in *Proceedings of the 22nd International Society for Music Information Retrieval Conference*, 2021. [Online]. Available: https://archives.ismir.net/ismir2021/paper/000058.pdf

[37] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[38] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization 3rd int," in *Conf. for Learning Representations, San*, 2014.

[39] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in neural information processing systems*, vol. 30, 2017.

[40] M. F. Naeem, S. J. Oh, Y. Uh, Y. Choi, and J. Yoo, "Reliable fidelity and diversity metrics for generative models," in *International Conference on Machine Learning*. PMLR, 2020, pp. 7176–7185.

[41] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 723–773, 2012.

[42] B. A. y Arcas, "Do Large Language Models Understand Us?" *Daedalus*, vol. 151, no. 2, pp. 183–197, 05 2022. [Online]. Available: https://doi.org/10.1162/daed_a_01909

[43] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler *et al.*, "Emergent abilities of large language models," *Transactions on Machine Learning Research (TMLR)*, 2022.

[44] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International Conference on Machine Learning*. PMLR, 2015, pp. 2256–2265.

[45] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[46] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," in *ICLR*, 2021.

[47] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[48] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," *University of Montreal*, vol. 1341, no. 3, p. 1, 2009.

[49] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[50] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *stat*, vol. 1050, p. 1, 2014.

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[52] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 976–11 986.

[53] S. Woo, S. Debnath, R. Hu, X. Chen, Z. Liu, I. S. Kweon, and S. Xie, "Convnext v2: Co-designing and scaling convnets with masked autoencoders," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 16 133–16 142.

[54] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[55] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[56] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[57] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[58] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[59] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[60] H. Wang, Y. Zou, H. Cheng, and L. Ye, "Diffuseroll: multi-track multi-attribute music generation based on diffusion model," *Multimedia Systems*, vol. 30, no. 1, p. 19, 2024.

[61] G. Mittal, J. Engel, C. Hawthorne, and I. Simon, "Symbolic music generation with diffusion models," *arXiv preprint arXiv:2103.16091*, 2021.

[62] J. A. Lupker, "Score-transformer: A deep learning aid for music composition," in *NIME 2021*. PubPub, 2021.

[63] T. Baoueb, H. Liu, M. Fontaine, J. Le Roux, and G. Richard, "Specdiff-gan: A spectrally-shaped noise diffusion gan for speech and music synthesis," in *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2024, pp. 986–990.

[64] C. Hawthorne, I. Simon, A. Roberts, N. Zeghidour, J. Gardner, E. Manilow, and J. Engel, "Multi-instrument music synthesis with spectrogram diffusion," *arXiv preprint arXiv:2206.05408*, 2022.

[65] J. Copet, F. Kreuk, I. Gat, T. Remez, D. Kant, G. Synnaeve, Y. Adi, and A. Défossez, "Simple and controllable music generation," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[66] A. Agostinelli, T. I. Denk, Z. Borsos, J. Engel, M. Verzetti, A. Caillon, Q. Huang, A. Jansen, A. Roberts, M. Tagliasacchi *et al.*, "Musiclm: Generating music from text,"

arXiv preprint arXiv:2301.11325, 2023.

[67] J. Engel, L. Hantrakul, C. Gu, and A. Roberts, "Ddsp: Differentiable digital signal processing," *arXiv preprint arXiv:2001.04643*, 2020.

[68] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, I. Simon, C. Hawthorne, N. Shazeer, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer: Generating music with long-term structure," in *International Conference on Learning Representations*, 2018.

[69] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," *arXiv preprint arXiv:2005.00341*, 2020.

[70] S. Narasimhaswamy, U. Bhattacharya, X. Chen, I. Dasgupta, S. Mitra, and M. Hoai, "Handiffuser: Text-to-image generation with realistic hand appearances," *arXiv preprint arXiv:2403.01693*, 2024.

[71] J. Betker, G. Goh, L. Jing, T. Brooks, J. Wang, L. Li, L. Ouyang, J. Zhuang, J. Lee, Y. Guo *et al.*, "Improving image generation with better captions," *Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf*, vol. 2, no. 3, p. 8, 2023.

[72] A. Y. J. Ha, J. Passananti, R. Bhaskar, S. Shan, R. Southen, H. Zheng, and B. Y. Zhao, "Organic or diffused: Can we distinguish human art from ai-generated images?" *arXiv preprint arXiv:2402.03214*, 2024.

[73] X. Dai, J. Hou, C.-Y. Ma, S. Tsai, J. Wang, R. Wang, P. Zhang, S. Vandenhende, X. Wang, A. Dubey *et al.*, "Emu: Enhancing image generation models using photogenic needles in a haystack," *arXiv preprint arXiv:2309.15807*, 2023.

[74] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel *et al.*, "Scaling rectified flow transformers for high-resolution image synthesis," in *Forty-first International Conference on Machine Learning*, 2024.

[75] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[76] N. Zeghidour, A. Luebs, A. Omran, J. Skoglund, and M. Tagliasacchi, "Soundstream: An end-to-end neural audio codec," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 495–507, 2021.

[77] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu *et al.*, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, vol. 12, 2016.

[78] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[79] M. Nasr, N. Carlini, J. Hayase, M. Jagielski, A. F. Cooper, D. Ippolito, C. A. Choquette-Choo, E. Wallace, F. Tramèr, and K. Lee, "Scalable extraction of training data from (production) language models," *arXiv preprint arXiv:2311.17035*, 2023.

[80] J. Von Oswald, E. Niklasson, E. Randazzo, J. Sacramento, A. Mordvintsev, A. Zhmoginov, and M. Vladymyrov, "Transformers learn in-context by gradient descent," in *International Conference on Machine Learning*. PMLR, 2023, pp. 35 151–35 174.

[81] T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh, "Autoprompt: Eliciting knowledge from language models with automatically generated prompts," *arXiv preprint arXiv:2010.15980*, 2020.

[82] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," *arXiv preprint arXiv:2104.08691*, 2021.

[83] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[84] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. A. Raffel, "Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1950–1965, 2022.

[85] N. Hyeon-Woo, M. Ye-Bin, and T.-H. Oh, "Fedpara: Low-rank hadamard product for communication-efficient federated learning," *arXiv preprint arXiv:2108.06098*, 2021.

[86] R. Agarwal, A. Singh, L. M. Zhang, B. Bohnet, S. Chan, A. Anand, Z. Abbas, A. Nova, J. D. Co-Reyes, E. Chu *et al.*, "Many-shot in-context learning," *arXiv preprint arXiv:2404.11018*, 2024.

[87] A. Bertsch, M. Ivgi, U. Alon, J. Berant, M. R. Gormley, and G. Neubig, "In-context learning with long-context models: An in-depth exploration," *arXiv preprint arXiv:2405.00200*, 2024.

[88] L. Schulze Balhorn, J. M. Weber, S. Buijsman, J. R. Hildebrandt, M. Ziefle, and A. M. Schweidtmann, "Empirical assessment of chatgpt's answering capabilities in natural science and engineering," *Scientific Reports*, vol. 14, no. 1, p. 4998, 2024.

[89] T. Webb, K. J. Holyoak, and H. Lu, "Emergent analogical reasoning in large language models," *Nature Human Behaviour*, vol. 7, no. 9, pp. 1526–1541, 2023.

[90] X. Fang, W. Xu, F. A. Tan, J. Zhang, Z. Hu, Y. Qi, S. Nickleach, D. Socolinsky, S. Sengamedu, and C. Faloutsos, "Large language models on tabular data–a survey," *arXiv preprint arXiv:2402.17944*, 2024.

[91] J. Foote, "Visualizing music and audio using self-similarity," in *Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, 1999, pp. 77–80.

[92] J. P. Bello, "Measuring structural similarity in music," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 7, pp. 2013–2025, 2011.

[93] B. Martin, M. Robine, P. Hanna *et al.*, "Musical structure retrieval by aligning self-similarity matrices." in *ISMIR*, 2009, pp. 483–488.

[94] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in neural information processing systems*, vol. 30, 2017.

[95] F. Ribeiro, D. Florêncio, C. Zhang, and M. Seltzer, "Crowdmos: An approach for crowdsourcing mean opinion score studies," in *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2011, pp. 2416–2419.

[96] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[97] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen, "Large language models as optimizers," *arXiv preprint arXiv:2309.03409*, 2023.

[98] E. R. Miranda and M. M. Wanderley, *New digital musical instruments: control and interaction beyond the keyboard*. AR Editions, Inc., 2006, vol. 21.

[99] J. Sullivan and M. M. Wanderley, "Surveying digital musical instrument use across diverse communities of practice," *Computer Music Multidisciplinary Research (CMMR)*, p. 745, 2019.

[100] D. Schwarz, W. Liu, and F. Bevilacqua, "A survey on the use of 2d touch interfaces for musical expression," in *New Interfaces for Musical Expression (NIME)*, 2020.

[101] L. Atassi, "Allocentric and Egocentric Controllers: Similarities and Differences," *Leonardo*, vol. 55, no. 4, pp. 394–398, 08 2022.

[102] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2019.

[103] H. Hotelling, "Analysis of a complex of statistical variables into principal components." *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[104] H. Zou, T. Hastie, and R. Tibshirani, "Sparse principal component analysis," *Journal of computational and graphical statistics*, vol. 15, no. 2, pp. 265–286, 2006.

[105] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental learning for robust visual tracking," *International journal of computer vision*, vol. 77, no. 1, pp. 125–141, 2008.

[106] P. Cohen, S. G. West, and L. S. Aiken, *Applied multiple regression/correlation analysis for the behavioral sciences*. Psychology press, 2014.

[107] "Linear discriminant analysis - bit by bit," https://web.archive.org/web/20220913032330/https://sebastianraschka.com/Articles/2014_python_lda.html, accessed: 2022-09-10.

[108] L. Atassi, "Hinged t-sne for musical interfaces," 06 2022, p. NIME 2022.

[109] L. McInnes, J. Healy, N. Saul, and L. Großberger, "Umap: Uniform manifold approximation and projection," *Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018.

[110] L. van der Maaten, "Learning a parametric embedding by preserving local structure." *Journal of Machine Learning Research - Proceedings Track*, vol. 5, pp. 384–391, 01 2009.

[111] D. P. Kingma, M. Welling *et al.*, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019.

[112] "Compressionvae - github repository," https://github.com/maxfrenzel/CompressionVAE, accessed: 2022-09-10.

[113] E. Benjamin and J. Altosaar, "Musicmapper: interactive 2d representations of music samples for in-browser remixing and exploration." in *NIME*, 2015, pp. 325–326.

[114] "Infinite drum machine," https://github.com/googlecreativelab/aiexperiments-drum-machine, accessed: 2022-09-10.

[115] A. Lerch, *An introduction to audio content analysis: Applications in signal processing and music informatics*. Wiley-IEEE Press, 2012.

[116] L. Atassi, "Allocentric and egocentric controllers: Similarities and differences," *Leonardo*, vol. 55, no. 4, pp. 394–398, 2022.

[117] M. M. Wanderley and P. Depalle, "Gestural control of sound synthesis," *Proceedings of the IEEE*, vol. 92, no. 4, pp. 632–644, 2004.

[118] F. Morreale *et al.*, "Design for longevity: Ongoing use of instruments from nime 2010-14," 2017.

[119] F. Morreale, A. McPherson, M. Wanderley *et al.*, "Nime identity from the performer's perspective," 2018.

[120] F. Berthaut, C. Arslan, and L. Grisoni, "Revgest: Augmenting gestural musical instruments with revealed virtual objects," in *International Conference on New Interfaces for Musical Expression*, 2017.

[121] S. Sentürk, S. W. Lee, A. Sastry, A. Daruwalla, and G. Weinberg, "Crossole: A gestural interface for composition, improvisation and performance using kinect." in *NIME*. Citeseer, 2012.

[122] P. Dahlstedt and A. S. Dahlstedt, "Otokin: Mapping for sound space exploration through dance improvisation." in *NIME*, 2019, pp. 156–161.

[123] K. Ng, "Interactive gesture music performance interface," in *Proceedings of the 2002 conference on New interfaces for musical expression*, 2002, pp. 1–2.

[124] D. Wessel, M. Wright, and J. Schott, "Intimate musical control of computers with a variety of controllers and gesture mapping metaphors." in *NIME*, vol. 2, 2002, pp. 1–3.

[125] T. Meilinger and G. Vosgerau, "Putting egocentric and allocentric into perspective," in *Spatial Cognition VII: International Conference, Spatial Cognition 2010, Mt. Hood/Portland, OR, USA, August 15-19, 2010. Proceedings 7*. Springer, 2010, pp. 207–221.

[126] M. Mainsbridge and K. Beilharz, "Body as instrument–performing with gestural interfaces," in *Proceedings of the international conference on new interfaces for musical expression*, 2014.

[127] M. A. O. Pérez, B. Knapp, and M. Alcorn, "Diamair: composing for choir and integral music controller," in *Proceedings of the 7th international conference on New interfaces for musical expression*, 2007, pp. 289–292.

[128] T. Magnusson, "Designing constraints: Composing and performing with digital musical systems," *Computer music journal*, vol. 34, no. 4, pp. 62–73, 2010.

[129] K. Tahiroglu, M. Gurevich, and B. R. Knapp, "Contextualising idiomatic gestures in musical interactions with nimes," in *International Conference on New Interfaces for Musical Expression*. New Interfaces for Musical Expression (NIME), 2018.

[130] A. Hunt and R. Kirk, "Mapping strategies for musical performance," *Trends in gestural control of music*, vol. 21, no. 2000, pp. 231–258, 2000.

[131] A. Hunt and M. M. Wanderley, "Mapping performer parameters to synthesis engines," *Organised sound*, vol. 7, no. 2, pp. 97–108, 2002.

[132] A. Altavilla, B. Caramiaux, and A. Tanaka, "Towards gestural sonic affordances," 2013.

[133] P. Rojas, "To become one with the instrument: The unfolding of a musical topography," *Culture & Psychology*, vol. 21, no. 2, pp. 207–230, 2015.

[134] T. Magnusson and E. Hurtado, "The phenomenology of musical instruments: a survey," *eContact! Improv*, vol. 10, no. 4, 2008.

[135] J. V. Gomez, K. Tahiroglu, and J. Kildal, "Idiomatic composition practices for new musical instruments: Context, background and current applications," in *International Conference on New Interfaces for Musical Expression*. International Conference on New Interfaces for Musical Expression (NIME), 2017, pp. 174–179.

[136] "Sonami l (2021) lady's glove," https://sonami.net/portfolio/items/ladys-glove/, accessed: 2022-09-10.

[137] E. N. Jessop, "The vocal augmentation and manipulation prosthesis (vamp): A conducting-based gestural controller for vocal performance." in *NIME*, 2009, pp. 256–259.

[138] T. Mäki-Patola, J. Laitinen, A. Kanerva, and T. Takala, "Experiments with virtual reality instruments," in *Proceedings of the 2005 conference on New interfaces for musical expression*, 2005, pp. 11–16.

[139] D. Brown, N. Renney, A. Stark, C. Nash, and T. Mitchell, "Leimu: Gloveless music interaction using a wrist mounted leap motion," *interaction*, vol. 1, no. 2, pp. 3–4, 2016.

[140] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[141] R. Arandjelovic and A. Zisserman, "Look, listen and learn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 609–617.