UNIVERSITY OF CALIFORNIA SAN DIEGO

Enabling New Musical Interactions with Machine Learning

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

in

Music

by

Chris Donahue

2019

DEDICATION

For my mother and father

EPIGRAPH

AI [alone] would have the capacity to write a good song,

but not a great one.

—Nick Cave

TABLE OF CONTENTS

Signature Page iii

Dedication iv

Epigraph v

Table of Contents vi

List of Figures x

List of Tables xv

Acknowledgements xvii

Vita xix

Abstract of the Dissertation xx

I Introduction 1

Chapter 1 Introduction 2
1.1 Dissertation organization 6

II Music generation with machine learning 7

Chapter 2 Preliminaries 8
2.1 Musical representations 9
2.2 Music as a factorized probability distribution 11
2.2.1 Discretizing $p(\text{music})$ 12
2.2.2 Factorizing $P(\text{music})$ 12
2.3 Simplifying and generating $P(\text{score})$ 14
2.3.1 Preserving sequential aspects of music in $P(\text{score})$ 16
2.3.2 Language models for $P(\text{score})$ 19

Chapter 3 Music generation in the symbolic domain 21
3.1 Related work 23
3.2 Dataset preliminaries 24
3.2.1 NES ensemble preliminaries 24
3.3 Representations and tasks 25
3.3.1 Blended score representation and task 26

3.3.2 Separated score representation and task 26
3.3.3 Event-based representation and task 27
3.3.4 Expressive performance generation 29
3.4 Methodology 30
3.4.1 Transformer architecture 30
3.4.2 Pre-training 31
3.5 Event-based score generation experiments 33
3.5.1 Data augmentation and pre-training 34
3.5.2 Baselines 35
3.5.3 Quantitative analysis 35
3.6 Expressive performance generation experiments 36
3.7 User study 39
3.7.1 Turing test 39
3.7.2 Preference test 41
3.8 Pairing LakhNES with humans 43
3.9 Conclusion 43
3.10 Acknowledgements 43

Chapter 4 Music generation in the acoustic domain 45
4.1 GAN Preliminaries 47
4.2 WaveGAN 48
4.2.1 Intrinsic differences between audio and images 48
4.2.2 WaveGAN architecture 49
4.3 Understanding and mitigating artifacts in generated audio 51
4.3.1 Learned post-processing filters 52
4.3.2 Upsampling procedure 53
4.3.3 Phase shuffle 54
4.4 SpecGAN: Generating semi-invertible spectrograms 55
4.5 Experimental protocol 56
4.6 Evaluation methodology 58
4.6.1 Inception score 58
4.6.2 Nearest neighbor comparisons 59
4.6.3 Qualitative human judgements 60
4.7 Results and discussion 60
4.8 Related work 62
4.9 Improving SpecGAN with adversarial vocoding 64
4.9.1 Summary of contributions 66
4.10 Audio feature preliminaries 66
4.11 Measuring the effect of magnitude and phase estimation on speech naturalness 68
4.12 Adversarial vocoding 69
4.12.1 Network architecture 71
4.13 Experiments 73

4.13.1 Vocoding LJ Speech mel spectrograms 74
4.13.2 Unsupervised audio synthesis 76
4.14 Conclusion 77
4.15 Acknowledgements 77

III New musical interactions 79

Chapter 5 Piano Genie 81
5.1 Introduction 81
5.2 Related Work 84
5.3 Methods 84
5.4 Experiments and Analysis 86
5.4.1 Analysis 88
5.5 User Study 91
5.6 Web demo details 92
5.7 Conclusion 93
5.8 Acknowledgements 94

Chapter 6 Dance Dance Convolution 95
6.1 Introduction 95
6.1.1 Contributions 99
6.2 Data 100
6.3 Problem Definition 101
6.4 Methods 103
6.4.1 Audio Representation 103
6.4.2 Step Placement 104
6.4.3 Peak Picking 106
6.4.4 Step Selection 107
6.5 Experiments 109
6.5.1 Step Placement 109
6.5.2 Step Selection 111
6.6 Discussion 112
6.7 Related Work 114
6.8 Conclusions 116
6.9 Acknowledgements 116

Chapter 7 Other interactive systems 117
7.1 DrumGAN 117
7.2 Neural Loops 119
7.3 Semantically-decomposed GANs 121

IV Conclusion 124

Chapter 8 Conclusion 125

LIST OF FIGURES

Figure 1.1: Piano improvisation is a task hitherto inaccessible by non-musicians despite many having the potential capacity. This task combines high-level musical intuition with lower-level planning and mechanical expertise. Could we relax (Chapter 5) or one day circumvent (dashed line) the intermediate steps which currently prevent non-musicians from expressing their dormant musical intuition through the piano? 3

Figure 2.1: Symbolic (left) and acoustic (right) representations of the same piece of music (*Prelude in C major, BWV 846, J. S. Bach*). **Sheet music** is a symbolic format most familiar to musicians. **MIDI** stores all of the information about a composition in a compact list of musical events. **Piano roll** encodes MIDI information into a time-pitch grid retaining the relationships between notes. **Waveforms** (recordings) are a digital representation of a musical pressure wave. **Spectrograms** are a time-frequency decomposition of waveforms which resemble how the human auditory system processes sound that arrives at our ears. 10

Figure 2.2: Comparing the pitch class frequencies for specific pieces composed by Bach and Schönberg. 14

Figure 2.3: Pitch class transition likelihoods across all of Bach's *Well-Tempered Clavier* (BWV 846–893). The transitions reflect our understanding of Bach's compositional style. For example, major seconds are the most likely intervals while tritones are the least likely. 17

Figure 2.4: Trigram, 4-gram, and 5-gram models for Bach's *Well-Tempered Clavier*. While the exponential growth of the model parameters make it difficult to interpret the distributions visually, we can see that the sparsity of the models (how many *n*-grams were *not* present in the corpus) increases as the context grows (i.e., there is more whitespace in the diagram). 18

Figure 2.5: Comparing likelihoods assigned to sequences of five pitch classes by a 5-gram distribution (left) and a recurrent neural network (right) trained on the same corpus. The recurrent neural network has learned a smooth parameterization of the distribution that allows us to estimate reasonable likelihoods for sequences which did not appear in the training data. 19

Figure 3.1: Four representations for a segment of *Ending Theme* from *Abadox* (1989) by composer Kiyohiro Sada. The blended score (fig. 3.1a), used in prior polyphonic generation research, is degenerate when multiple voices play the same note. 25

Figure 3.2: A visual comparison between the piano roll representation of the original NES-MDB paper [Donahue et al., 2018c] (**top**) and the event representation of this work (**bottom**). In the piano roll representation, the majority of information is the same across timesteps. In our event representation, each timestep encodes a musically-meaningful change. 28

Figure 3.3: Illustration of our mapping heuristic used to enable transfer learning from Lakh MIDI to NES-MDB. We identify monophonic instruments from the arbitrary ensembles in Lakh MIDI and randomly assign them to the fixed four-instrument ensemble of NES-MDB. 32

Figure 3.4: LSTM Note+Auto expressive performance model that observes both the score and its prior output. 38

Figure 3.5: Measuring the performance improvement when doubling the amount of Lakh MIDI pre-training before fine-tuning Transformer-XL on NES-MDB. Each datapoint represents the result of a fine-tuning run starting from 0, 1, 2, or 4 epochs of Lakh MIDI pre-training. Additional amounts of pre-training appear to improve performance, though with diminishing returns. 40

Figure 3.6: Human accuracy at distinguishing computer-generated examples from human-composed ones (error bars are standard error). Users were presented with pairs of clips (one human, one computer) and tasked with identifying which was composed by a human. Random examples are used as a control and we filtered annotators with accuracy less than 1 on those pairs. A *lower* accuracy is better as it indicates that the annotators confused a particular model with the real data more often. 41

Figure 3.7: Proportion of comparisons where humans preferred an example from each model over an example from another random model (error bars are standard error). Users were presented with pairs of clips from different methods and asked which they preferred. Pairs of random data and human-composed clips are used as a control and we filtered annotators who preferred random. A *higher* ratio is better as it indicates that the annotators preferred results from that method more often than another. 42

Figure 4.1: First eight principal components for 5x5 patches from natural images (**left**) versus those of length-25 audio slices from speech (**right**). Periodic patterns are unusual in natural images but a fundamental structure in audio. 49

Figure 4.2: Depiction of the transposed convolution operation for the first layers of the DCGAN [Radford et al., 2016] (**left**) and WaveGAN (**right**) generators. DCGAN uses small (5x5), two-dimensional filters while WaveGAN uses longer (length-25), one-dimensional filters and a larger upsampling factor. Both strategies have the same number of parameters and numerical operations. 50

Figure 4.3: (**Top**): Average impulse response for 1000 random initializations of the WaveGAN generator. (**Bottom**): Response of learned post-processing filters for speech and bird vocalizations. Post-processing filters reject frequencies corresponding to noise byproducts created by the generative procedure (top). The filter for speech boosts signal in prominent speech bands, while the filter for bird vocalizations (which are more uniformly-distributed in frequency) simply reduces noise presence. 52
Figure 4.4: Depiction of the upsampling strategy used by transposed convolution (zero insertion) and other strategies which mitigate aliasing: nearest neighbor, linear and cubic interpolation. 53
Figure 4.5: At each layer of the WaveGAN discriminator, the phase shuffle operation perturbs the phase of each feature map by Uniform $\sim [-n,n]$ samples, filling in the missing samples (dashed outlines) by reflection. Here we depict all possible outcomes for a layer with four feature maps ($n = 1$). 54
Figure 4.6: **Top**: Random samples from each of the five datasets used in this study, illustrating the wide variety of spectral characteristics. **Middle**: Random samples generated by WaveGAN for each domain. WaveGAN operates in the time domain but results are displayed here in the frequency domain for visual comparison. **Bottom**: Random samples generated by SpecGAN for each domain. 56
Figure 4.7: Depiction of stages in common audio feature extraction pipelines and corresponding inversion. The two obstacles to vocoding are (1) estimating linear-frequency magnitude spectra from log-frequency mel spectra (outlined in green dashed line), and (2) estimating phase information from magnitude spectra (outlined in blue dotted line). We focus on magnitude estimation in this paper, observing that coupling an ideal solution to this subproblem with a phase estimation heuristic can produce high-quality speech (Table 4.2). . 67
Figure 4.8: Adversarial Vocoder Model: The generator performs an image-to-image translation from the estimated magnitude spectrogram to the actual magnitude spectrogram guided by an adversarial loss from the discriminator and the L_1 distance between the generated and actual magnitude spectrogram 72

Figure 5.1: Using Piano Genie to improvise on a Disklavier (motorized piano) via MIDI. 82
Figure 5.2: Piano Genie consists of a discrete sequential autoencoder. A bidirectional RNN encodes discrete piano sequences (88 keys) into smaller discrete latent variables (8 "buttons"). The unidirectional decoder is trained to map the latents back to piano sequences. During inference, the encoder is replaced by a human improvising on buttons. 83

Figure 5.3: Comparison of the quantization scheme for two autoencoder strategies with discrete latent spaces. The VQ-VAE (left) learns the positions of k centroids in a d-dimensional embedding space (in this figure $k = 8, d = 2$, in our experiments $k = 8, d = 4$). An encoder output vector (grey circle) is quantized to its nearest centroid (yellow circle) before decoding. Our IQAE strategy (right) quantizes a scalar encoder output (grey) to its nearest neighbor (yellow) among $k = 8$ centroids evenly spaced between -1 and 1. 85
Figure 5.4: Qualitative comparison of the 8-button encodings for a given melody (top) by the VQ-VAE (middle) and our IQAE with L_{contour} (bottom). Horizontal is note index. The encoding learned by the IQAE echoes the contour of the musical input. 89
Figure 5.5: A user engages with the Piano Genie web interface during our user study. . 90

Figure 6.1: Proposed *learning to choreograph* pipeline for four seconds of the song *Knife Party feat. Mistajam - Sleaze*. The pipeline ingests audio features (**Bottom**) and produces a playable DDR choreography (**Top**) corresponding to the audio. 96
Figure 6.2: A four-beat measure of a typical chart and its rhythm depicted in musical notation. **Red:** quarter notes, **Blue:** eighth notes, **Yellow:** sixteenth notes, **(A)**: *jump* step, **(B)**: *freeze* step 98
Figure 6.3: Five seconds of choreography by difficulty level for the song *KOAN Sound - The Edge* from the Fraxtil training set. 100
Figure 6.4: Number of steps per rhythmic subdivision by difficulty in the Fraxtil dataset. 102
Figure 6.5: C-LSTM model used for step placement 104
Figure 6.6: One second of peak picking. **Green:** Ground truth region **(A)**: true positive, **(B)**: false positive, **(C)**: false negative, **(D)**: two peaks smoothed to one by Hamming window, **(E)**: misaligned peak accepted as true positive by $\pm 20ms$ tolerance 107
Figure 6.7: LSTM model used for step selection 108
Figure 6.8: **Top:** A real step chart from the *Fraxtil* dataset on the song *Anamanaguchi - Mess*. **Middle:** One-step lookahead *predictions* for the LSTM model, given Fraxtil's choreography as input. The model predicts the next step with high accuracy (errors in red). **Bottom:** Choreography generated by conditional LSTM model. 113

Figure 7.1: Interface for a demo of DrumGAN, which combines a WaveGAN generative model of drum one-shots with a traditional step sequencer. 118
Figure 7.2: The interface for *Neural Loops*, which combines multiple neural network generative models with a traditional musical sequencer to produce an interactive tool for exploring four-bar musical trios. 120
Figure 7.3: Schematic diagram for *Neural Loops* illustrating the generative models used as part of a factorized approach at modeling four-bar musical waveforms. . 121

Figure 7.4: The factorized latent spaces of semantically-decomposed generative adversarial networks allow for independent manipulation of salient and contingent aspects of multimedia such as face portraits and visual artworks. 122

LIST OF TABLES

Table 3.1: Schematic for our event-based representation of NES-MDB, reminiscent of the one used in *Performance RNN* [Simon and Oore, 2017]. The 631 events in our representation are distributed among time-shift (ΔT) events (which allow for nuanced timing), and note off/on events for individual instruments (as in typical MIDI). 29

Table 3.2: Quantitative performance of various models trained on the event-based representation (631 event types) of NES-MDB. *Params* indicates the number of parameters of each model. *Epochs* is the number of data epochs the model observed before early stopping based on the validation data. *Test PPL* represents the perplexity of the model on the test data, i.e., the exponentiation of its average negative log-likelihood on the test data. A *lower* perplexity indicates that the model *better* fits this unseen data. 35

Table 3.3: Results for expressive performance experiments evaluated at points of interest (POI). Results are broken down by expression category (e.g. V_{NO} is noise velocity, T_{P1} is pulse 1 timbre) and aggregated at POIs and globally (All). . . 37

Table 4.1: Quantitative and qualitative (human study) results for SC09 experiments comparing real and generated data. A higher inception score suggests that semantic modes of the real data distribution have been captured. $|D|_{\text{self}}$ indicates the intra-dataset diversity relative to that of the real test data. $|D|_{\text{train}}$ indicates the distance between the dataset and the training set relative to that of the test data; a low value indicates a generative model that is overfit to the training data. Acc. is the overall accuracy of humans on the task of labeling class-balanced digits (random chance is 0.1). Sound *quality*, *ease* of intelligibility and speaker *diversity* are mean opinion scores (1-5); higher is better. 61

Table 4.2: Ablating the effect of heuristics for magnitude and phase estimation on mean opinion score (MOS) of speech naturalness with 95% confidence intervals. **Bolded** entries show that coupling an ideal solution to either subproblem (real data used as a proxy) with a good heuristic for the other yields speech with only 2–9% lower MOS than real speech ($p < 0.05$). 69

Table 4.3: Comparison of vocoding methods on mel spectrograms with 80 bins. We display comparative mean opinion scores from two separate user studies for vocoding spectrograms extracted from real speech (MOS-Real) and spectrograms generated by a state-of-the-art TTS method (MOS-TTS) with 95% confidence intervals. $\times$ RT denotes the speed up over real time; higher is faster. MB denotes the size of each model in megabytes. 75

Table 4.4: Comparison of heuristic and adversarial vocoding of spectrograms with different levels of mel compression. Adversarial vocoding can vocode highly compressed mel spectrograms with relatively less drop in speech naturalness as compared to a heuristic. 75

Table 4.5: Combining our adversarial vocoding approach with GAN-generated mel spectrograms outperforms our prior work in unsupervised generation of individual words by all metrics. 76

Table 5.1: Quantitative results comparing an RNN language model, the VQ-VAE ($k = 8, d = 4$), and our proposed IQAE model ($k = 8$) with and without contour regularization. ΔT adds time shift features to model. PPL is perplexity: $e^{L_{\text{recons}}}$. CVR is contour violation ratio: the proportion of timesteps where the sign of the melodic interval $\neq$ that of the button interval. Gold is the mean squared error in button space between the encoder outputs for familiar melodies and manually-created gold standard button sequences for those melodies. Lower is better for all metrics. 87

Table 5.2: Results for a small ($n = 8$) user study for Piano Genie. Partipicants were given up to three minutes to improvise on three possible mappings between eight buttons and an 88-key piano: 1) (*G-maj*) the eight buttons are mapped to a G-major scale, 2 (*language model*) all buttons trigger our baseline language model, 3 (*Piano Genie*) our proposed method. (*Time*) is the average amount of time in seconds that users improvised with a mapping. (*Per.*, *Mus.*, *Con.*) are the respective averages users expressed for enjoyment of performance experience, enjoyment of music, and level of control. 90

Table 6.1: Dataset statistics 101

Table 6.2: Results for step placement experiments 109

Table 6.3: Results for step selection experiments 110

ACKNOWLEDGEMENTS

Thanks to my Committee Chair Miller Puckette for his unconditional support throughout my PhD. His input was central to the success of my research projects despite several of them being outside his primary area of expertise. It is impossible to overstate my indebtedness to my Co-chair Julian McAuley, who offered me a level of guidance far beyond the call of a faculty member and molded me from a tinkerer into a researcher. Additional thanks to all of the members of my committee for their support and encouragement throughout my postgraduate studies.

I am indebted to Zachary C. Lipton, who guided me with remarkable patience through the process of publishing my first machine learning papers. Thanks to Nicolas Boulanger-Lewandowski at Google, who accepted me as an intern despite my inexperience, and set me along my current research path. Thanks to Stephen Burnaman and Terry McKinney for teaching me everything I know about musicianship. Thanks to Russell Pinkston, Ethan Greene, and Peter Stone who got me started on all of this computer music business.

My deepest thanks to my parents, Linda and Jim, for their continued love and support—I am incredibly lucky to have been raised by them. Thanks to my brother Jeff for his moral support and research guidance. Thanks to Maggie Henderson for being a constant source of respite during this rollercoaster ride known as a "PhD" and beyond.

Thank you to all of my friends here in San Diego and across the globe who have kept me sane throughout the process (to name a few): Abhay Agarwal, Adam Sydnor, Alex Forsythe, Andrew Elrod, Annie Hui-Hsin Hsieh, Ashvin Bashyam, Carolyn Yuan, Celeste Oram, Cheng-i Wang, Colin Zyskowski, Collin Buchan, Eric Lyday, Gabe Garcia, Jayhee Min, Jeff Sun, Jen Hsu, Johannes Regnier, Keir GoGwilt, Kevin Cannon, Kevin Haywood, Nick Kubala, Niko Millias, Paulina Hensman, Rachel Cassidy, Ryan Done, Ryan Sema, and Sam Asinof.

Thank you to all of my co-authors whose input and guidance was crucial to my research success to date (in chronological order): Tom Erbe, Miller Puckette, Zachary C. Lipton, Julian McAuley, Bo Li, Rohit Prabhavalkar, Huanru Henry Mao, Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Adam Roberts, Ian Simon, Sander Dieleman, Paarth Neekhara, Shlomo Dubnov, Yiting Ethan Li, and Garrison W. Cottrell.

Chapter 3 contains material found in the following two papers. (1) *The NES Music Database: A multi-instrumental dataset with expressive performance attributes*. 2018. Mao, Huanru Henry; McAuley, Julian. International Society for Music Information Retrieval 2018. (2) *LakhNES: Improving multi-instrumental music generation with cross-domain pre-training*. 2019. Mao, Huanru Henry; Li, Yiting Ethan; Cottrell, Garrison W.; McAuley, Julian. Currently under review for publication. The dissertation/thesis author was the primary investigator and author of these papers.

Chapter 4 contains material found in the following two papers. (1) *Adversarial audio synthesis*. 2018. McAuley, Julian; Puckette, Miller. International Conference on Learning Representations 2019. (2) *Expediting TTS synthesis with adversarial vocoding*. 2019. Neekhara, Paarth; Puckette, Miller; Dubnov, Shlomo; McAuley, Julian. Currently under review for publication. The dissertation/thesis author was the primary investigator and author of these papers.

Chapter 5, in full, is a reprint of the material as it appears in *Piano Genie*. 2019. Simon, Ian; Dieleman, Sander. ACM conference on Intelligent User Interfaces 2019. The dissertation/thesis author was the primary investigator and author of this paper.

Chapter 6, in full, is a reprint of the material as it appears in *Dance Dance Convolution*. 2017. Lipton, Zachary C.; McAuley, Julian. International Conference on Machine Learning 2017. The dissertation/thesis author was the primary investigator and author of this paper.

VITA

2013	B. S. in Computer Science, The University of Texas at Austin
2016	M. A. in Music, University of California San Diego
2019	Ph. D. in Music, University of California San Diego

PUBLICATIONS

Chris Donahue, Ian Simon, Sander Dieleman. “Piano Genie”, *ACM Conference on Intelligent User Interfaces*, 2019.

Chris Donahue, Julian McAuley, Miller Puckette. “Adversarial audio synthesis”, *International Conference on Learning Representations*, 2019.

Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, Adam Roberts. “GANSynth: Adversarial neural audio synthesis”, *International Conference on Learning Representations*, 2019.

Chris Donahue, Huanru Henry Mao, Julian McAuley. “The NES Music Database: A multi-instrumental dataset with expressive performance attributes”, *International Society for Music Information Retrieval*, 2018.

Chris Donahue, Zachary C. Lipton, Akshay Balsubramani, Julian McAuley. “Semantically decomposing the latent spaces of generative adversarial networks”, *International Conference on Learning Representations*, 2018.

Chris Donahue, Bo Li, Rohit Prabhavalkar. “Exploring speech enhancement with generative adversarial betworks for robust speech recognition”, *International Conference on Acoustics, Speech, and Signal Processing*, 2018.

Chris Donahue, Zachary C. Lipton, Julian McAuley. “Dance Dance Convolution”, *International Conference on Machine Learning*, 2017.

Chris Donahue, Tom Erbe, Miller Puckette. “Extended convolution techniques for cross-synthesis”, *International Computer Music Conference*, 2016.

ABSTRACT OF THE DISSERTATION

Enabling New Musical Interactions with Machine Learning

by

Chris Donahue

Doctor of Philosophy in Music

University of California San Diego, 2019

Miller Puckette, Chair
Julian McAuley, Co-Chair

Despite the ubiquity of computers leading to a steady increase in global music consumption, computing has yet to fundamentally transform the capacity for non-musicians to engage with music. The experience of non-musicians—that of passively listening to music—is largely the same as it was decades prior. In this dissertation, I describe my research which allows non-musicians and musicians alike to create, respond, modify, or otherwise *interact* with music in new ways. Examples include a system which allows non-musicians to improvise on musical instruments without experience, and a system which automatically choreographs music to provide an interactive listening and dancing experience.

Common to all of my work is the use of predictive models which allow us to efficiently sift through musical spaces to identify promising content. The identified content can then be further curated by humans as part of an interactive musical workflow. To build such predictive models, I use machine learning which seeks to extract and generalize patterns in human-composed music. Specifically, I focus on a subfield of machine learning called deep learning, which is capable of extracting such patterns in high-dimensional music representations including both symbolic and acoustic formats. My research focuses on both advancing the state of the art in deep learning for music (and other types of multimedia), and designing interfaces which allow humans to intuitively benefit from what deep learning systems have discovered about music.

Part I

Introduction

Chapter 1

Introduction

The primary goal of my research is to build systems which allow people to *interact* with music in new ways. By “interact”, I am referring broadly to any type of engagement with music apart from *passive listening*, i.e., normal listening experiences for musical recordings. New interactive systems could allow hitherto passive listeners to *actively* engage with or manipulate music they are listening to, or perhaps even create music themselves. In practice, my recent work has focused on augmenting the creative music capacity of humans by designing interfaces which allow them to pilot or curate generative music systems trained using machine learning. For non-musicians, interactive machine learning systems can lower the barrier to entry for music creation by allowing them to intuitively express their musical ideas. For musicians, such systems expedite laborious sub-tasks of digital music production (e.g. selecting a sample from a sample library) by designing interfaces which allow for efficient exploration.

Typical systems for creating music (e.g. musical instruments, sheet music, digital audio workstations) are complex and require high levels of musical expertise for desirable operation. Such expertise eludes the vast majority of the population because acquiring it requires substantial time and financial commitments. Accordingly, the prerequisite capacity and intuition for musical expression lies dormant in many non-musicians. A goal of my research is to build systems which

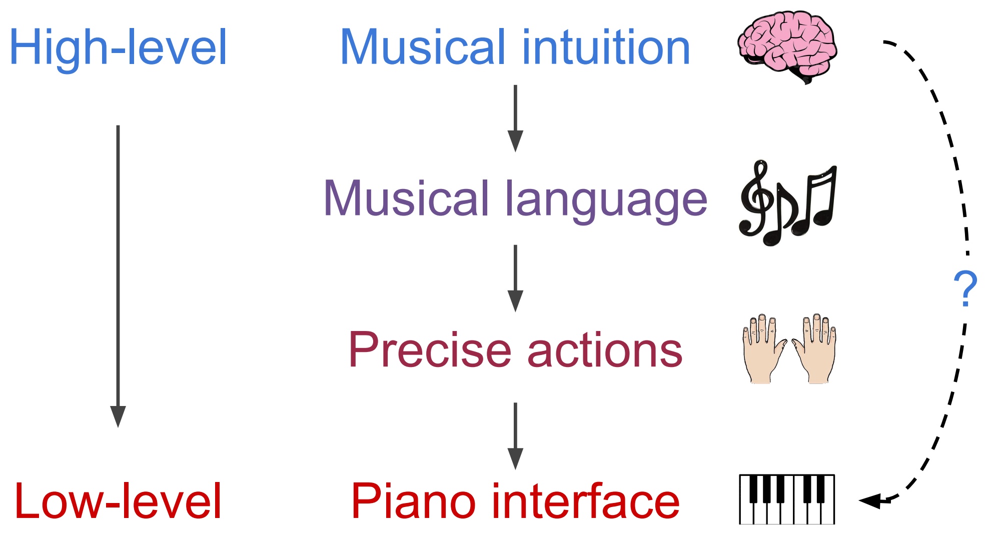


Figure 1.1: Piano improvisation is a task hitherto inaccessible by non-musicians despite many having the potential capacity. This task combines high-level musical intuition with lower-level planning and mechanical expertise. Could we relax (Chapter 5) or one day circumvent (dashed line) the intermediate steps which currently prevent non-musicians from expressing their dormant musical intuition through the piano?

allow for intuitive expression of musical ideas but do not require traditional forms of musical expertise.

To unpack the obstacles to building such systems, let's examine the practice of improvising on the piano. This task can be oversimplified into the following procedure (Figure 1.1). First, the improviser must mentally formulate their piece using their *musical intuition*. Then, perhaps implicitly, these ideas are translated into the discrete "language" of music, i.e., sequences of rhythms and pitches. Finally, this musical language must be translated into a precise series of physical actions that the performer inputs into the piano interface in real-time.

Arguably, much of the beauty and higher-level reasoning involved in piano improvisation resides in the first step of conceiving a piece from one's musical intuition. However, a disproportionate amount of the effort involved in *learning* to play the piano is on building the lower-level planning and dexterity required to precisely impart those ideas onto the keys. Of course, years of formal piano instruction also help to shape one's high-level musical intuition. But could we possibly decouple the development of these high-level skills from the low-level skills required to precisely input musical patterns on particular instruments? Could we create systems which circumvent these intermediate steps (dashed line Figure 1.1), allowing performers to translate their musical ideas *directly* into the piano? Such systems would disentangle musical intuition from mechanical expertise, providing a means for people to express the former without learning the latter. Directly translating signals from the brain into the piano remains science fiction, however later in this dissertation I will discuss a physical interface which achieves this high-level goal to a degree.

A separate issue with modern music creation systems (e.g. digital audio workstations) is that many common tasks are extremely laborious. For example, an electronic music producer might spend large amounts of time scouring endless directories of audio clips in search of the perfect kick drum sound for her track. She might also spend inordinate amounts of time manually adjusting the volume and timing of a drum pattern to achieve a desired effect. Can we design interfaces which allow musicians to quickly explore a trove of reasonable outcomes for such routine tasks? By transforming laborious manual tasks into efficient exercises in exploration and curation, such systems would augment the productivity of existing musicians.

All of the systems I have described so far involve making predictions on behalf of humans. As such, they require *predictive models* that can make musical decisions similarly to human experts. In my research, I use machine learning to build such models. Machine learning systems extract patterns from human-composed music which can then be used to make predictions for new data and offer state-of-the-art performance for a wide variety of musical prediction tasks. I will detail systems I built which can synthesize attributes of music in a variety of different contexts, from simpler tasks like creating expressive dynamics to more complex tasks like generating waveforms.

Machine learning (ML) systems for music suffer two primary drawbacks. The first is that the tooling infrastructure for training such systems is highly inaccessible, requiring extensive programming experience and heavy usage of the command line. Fortunately—with substantial effort—it is possible to package models trained in this unfriendly environment into familiar musical interfaces complete with knobs and faders. The second drawback is that ML systems are (currently) frail and largely incapable of high-level musical decision making. Luckily, it is possible through human-computer interfaces to pair the high-level planning abilities of humans (e.g. writing a chord progression for a song) with the efficiency of ML systems for low-level tasks (e.g. producing hundreds of reasonable melodies given a chord progression). Hence, a sizable portion of my research involves building and evaluating interfaces between people and predictive music systems.

At this point I hope to preemptively address a common criticism of my work. Namely, that building systems which lower the barrier to entry for music creation will cheapen the experience of existing musicians. This is not the case as existing musicians will always have the advantage of stronger musical intuition acquired through years of practice. Hence, musicians will always have higher capacity for precise creative expression regardless of the musical tooling available.

As we enter the era of artificially intelligent (AI) music systems, I believe strongly that humans *must* be kept in the loop. Beyond the aforementioned practical rationale, there are many other reasons why this is the case. I struggle to envision a future where computers are writing music for other computers, so human ears will continue to be the intended recipients. Hence, humans will always be the toughest critic and the only real ground truth for evaluating AI music systems. Additionally, no matter how compelling the music produced by an AI system is, ultimately I believe that humans will always have the advantage when it comes to true creativity, rather than just mimicking that which came before. My vision of AI is that, like electronics before it, it becomes yet another tool in our musical toolbox allowing for deeper musical expression.

1.1 Dissertation organization

I begin this dissertation with an extended discussion in Part II of my work on using machine learning to build generative models of music at both the symbolic and acoustic level. Chapter 2 provides an explanation of modeling music as a probability distribution, which will be central to the developments of the rest of the thesis. Chapter 3 will discuss the generation of discrete musical scores (i.e. symbolic representations of music), and Chapter 4 will discuss the generation of musical (and other types of) waveforms (i.e. acoustic representations of music).

In Part III, I describe my research on building systems which allow for new musical interactions with a focus on co-creation between humans and generative models of music. Chapter 5 will discuss my work on lowering the barrier to entry for playing musical instruments with machine learning. Chapter 6 will present a system which allows non-musicians to interact with their personal music collection through a dance-based video game. Chapter 7 contains information about other demos I have built which merge traditional music interfaces such as sequencers with powerful generative models of music. In Part IV, I offer some concluding remarks.

Part II

Music generation with machine learning

Chapter 2

Preliminaries

In this chapter I describe methodology that I have developed for building *music generation* systems with machine learning. As previously stated, my ultimate goal is to build machine learning systems which allow humans to interact with music in new ways. However, in this portion of my dissertation I focus exclusively on the machine learning methodology I have developed towards this goal, and discuss how humans can benefit from this work in Part III.

Machine learning (ML) systems extract patterns from collections of natural data in order to make predictions about the natural world. ML is most appropriate for tasks where it would be difficult to manually program a function to accomplish a particular goal. For example, it would be appropriate to use ML to recognize chord progressions from recordings of music. However, it would be less appropriate to use ML to identify a key signature from MIDI data, as a simple heuristic based on the prevalance of particular pitch classes would suffice.

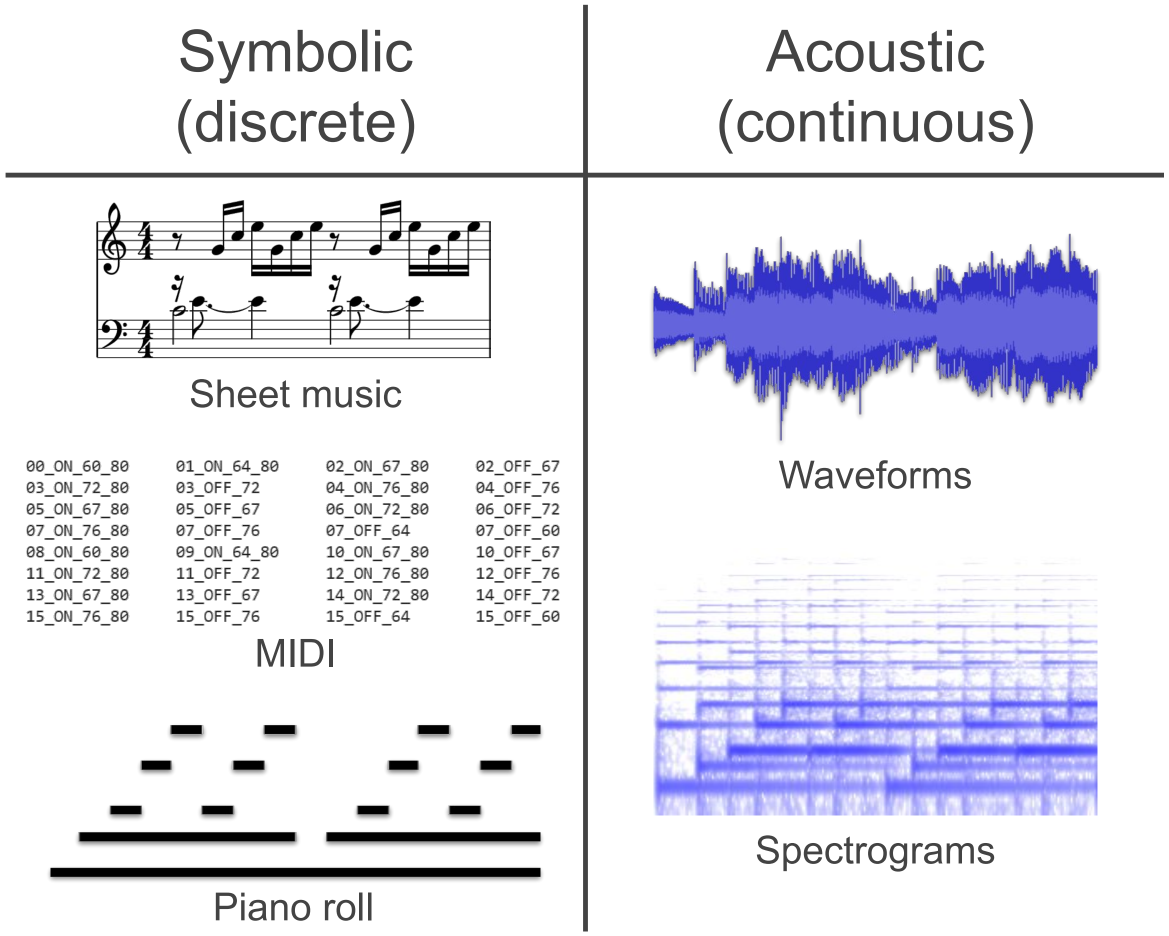
A particular area of machine learning called *deep learning* has recently been increasing in popularity because of its ability to model and make predictions about high-dimensional data [LeCun et al., 2015]. Music is inherently high-dimensional data with rich structure; accordingly, deep learning is my tool of choice for the predictive tasks I encounter in my research. The goal of deep learning is to train neural networks—non-linear functions often with millions of parameters—to map data from one domain to another. In a typical setting, a neural network is trained to perform a *supervised* learning task, e.g. labeling data. To accomplish this, data is passed into the neural network which returns estimates for the likelihood of particular labels. These estimates are compared to the real labels, and the error of the estimates is used to improve the parameters of the neural network, in a process known as *backpropagation* [Rumelhart et al., 1985].

Unlike in other domains such as image and speech recognition where deep learning dominates, labeled data is a scarce resource in music. Additionally, the objectives in applying machine learning to music are often less clear than the objectives of applying machine learning to e.g. speech recognition. Hence, in my research I also explore *unsupervised* deep learning approaches, where the goal is to train neural networks to recognize patterns in unlabeled data. For example, could we recognize commonalities across the hundreds of four-voice chorales composed by J.S. Bach? Often, my goal with using unsupervised learning is to *generate* new music in the style of the training data, hence I refer to these approaches as *generative modeling*. Could we extrapolate the patterns we extracted from Bach's chorales to generate convincing chorales in his style?

Because I seek to *generate* music, the majority of Part II will focus on unsupervised learning strategies. In Chapter 3, I will discuss methodology for learning to generate music in the symbolic domain. In Chapter 4, I will discuss music generation in the acoustic domain. First, I offer an explanation of both musical representations and treating music as a probability distribution which will inform later methodology.

2.1 Musical representations

Of considerable importance to the development of machine learning methodology for music is the choice of *input representation*. Here, input representation refers to how music data is presented to a neural network. As a coarse taxonomy, music representations can be divided into



```
00_ON_60_80   01_ON_64_80   02_ON_67_80   02_OFF_67
03_ON_72_80   03_OFF_72     04_ON_76_80   04_OFF_76
05_ON_67_80   05_OFF_67     06_ON_72_80   06_OFF_72
07_ON_76_80   07_OFF_76     07_OFF_64     07_OFF_60
08_ON_60_80   09_ON_64_80   10_ON_67_80   10_OFF_67
11_ON_72_80   11_OFF_72     12_ON_76_80   12_OFF_76
13_ON_67_80   13_OFF_67     14_ON_72_80   14_OFF_72
15_ON_76_80   15_OFF_76     15_OFF_64     15_OFF_60
```

Figure 2.1: Symbolic (left) and acoustic (right) representations of the same piece of music (*Prelude in C major, BWV 846, J. S. Bach*). **Sheet music** is a symbolic format most familiar to musicians. **MIDI** stores all of the information about a composition in a compact list of musical events. **Piano roll** encodes MIDI information into a time-pitch grid retaining the relationships between notes. **Waveforms** (recordings) are a digital representation of a musical pressure wave. **Spectrograms** are a time-frequency decomposition of waveforms which resemble how the human auditory system processes sound that arrives at our ears.

two categories: *symbolic* and *acoustic* (Figure 2.1). Symbolic representations are any strategy which represents music as discrete data, such that it could straightforwardly be engraved as sheet music. Common symbolic representations include MIDI and piano roll. Acoustic representations are any strategy which captures an audio waveform, such that it could be easily be played out of a loudspeaker. Common acoustic representations include time-domain waveforms and frequency-domain spectrograms.

The choice between symbolic and acoustic representations is one of many trade-offs. For example, we might intuitively gravitate towards symbolic music representations for music generation tasks, because they more compactly represent a composer's intentions than acoustic representations. Such a choice has its limitations however, as symbolic music data is far less common than acoustic music data. Hence, we might seek to generate music by modeling music in the audio domain, but this makes the task much more complicated because audio waveforms are high-dimensional and also entangle information about the underlying composition with information about a particular acoustic performance. Additionally, the choice of representation has implications for both the tasks that we can investigate and the types of neural networks that we can use to model the music in our chosen representation. Therefore, an input representation would be selected to match a particular task and methodology, or vice versa.

2.2 Music as a factorized probability distribution

An assumption that underlies all of the methodological contributions here is that music can be modeled as a probability distribution, and that any given piece of music can be viewed as a sample from this distribution. A probability distribution is a function that maps all possible configurations of sources of randomness (*random variables*) into a number which specifies how "likely" that configuration is. Henceforth I will refer to the hypothetical distribution of music as $p(\text{music}) : \text{music} \mapsto [0,1]$. Ideally, $p(\text{music})$ assigns equal likelihood to every piece of music, so that each of them would be sampled from the distribution with equal likelihood. Whether or not you agree philosophically with this definition, please consider it a conceit that we will use to build and understand generative models of music.

In this context, $p(\text{music})$ encapsulates all of the random factors that can affect a time-varying pressure signal which arrives at a human ear and is described by at least one person in the world as “music”. A random sample from this distribution is a real-valued, time-varying pressure signal with unbounded length. Never in my work do I attempt to directly model this probability distribution. Instead, I “fix” certain random factors to construct regions of this distribution which I can tractably model. First, I describe an important simplifying assumption.

2.2.1 Discretizing $p(\textbf{music})$

An important simplifying assumption I make is that music has been sampled at discrete points in time and its pressure (amplitude) has been quantized. I will refer to this distribution as $P(\text{music})$, where the capital letter is used to signify the discrete nature of samples from this probability distribution (as opposed to the continuous samples of $p(\text{music})$. Hence, a sample from this distribution $\boldsymbol{m} \in \mathbb{Z}^N \sim P(\text{music})$ is a vector of N (variable-length) samples with quantized amplitude values in set $\mathbb{Z}$ and a sampling frequency f_s. A single discrete sample $\boldsymbol{m} \sim P(\text{music})$ represents an infinite number of continuous samples from $p(\text{music})$. However, I make an assumption that humans would not be able to tell the difference between these infinite continuous samples given standard configurations of f_s and $\mathbb{Z}$ (e.g. 44.1kHz, 16-bit audio), and hence the discrete sample $\boldsymbol{m}$ is sufficient to represent them.

2.2.2 Factorizing $P(\textbf{music})$

$P(\text{music})$ encapsulates every source of randomness that can affect a digital musical waveform. To name a few possible sources: composer, performer, ensemble, instrument, microphone, temperature, composer's mood, length of time the ensemble played for, how much coffee the drummer had, etc. Hence, we can *factorize* this distribution by fixing certain random factors and modeling the remaining ones.

To determine which factorizations are reasonable, we turn to both our knowledge of how music arises naturally and the data we have available. For example, we know that some types of music arise from different performers playing the same score. Hence, we might factorize as $P(\text{music}) = P(\text{score}) \cdot P(\text{acoustic realization} \mid \text{score})$. This is a flexible approach that allows us to independently explore generative methods for music *score* and music *performance*. We might also only have data available for a specific composer, and thus by using their compositions as training data, we are implicitly modeling $P(\text{score} \mid \text{composer} = \text{Bach})$.

We know that an acoustic realization of a score also involves several factors. Hence, we simplify further to the following factorization

$$P(\text{music}) = P(\text{score}) \cdot P(\text{performance} \mid \text{score}) \cdot P(\text{acoustic} \mid \text{performance}). \tag{2.1}$$

In this factorization, *score* refers to all of the attributes necessary to engrave a piece of music as sheet music (Figure 2.1). *Performance* refers to all of the expressive performance attributes such as dynamics and rubato which a skilled performer uses when performing a rendition. Finally, *acoustic* refers to the acoustic properties that yield a waveform from a performance such as the timbre of the instruments, the room configuration, and the microphone used.

This factorization is convenient to work with because it allows us to examine each portion of this generative music "pipeline" independently. We can develop, for example, mix-and-match various methods of generating scores with methods for generating performances from scores. Throughout the remainder of Part II, I will discuss methods that pertain to the various components of Equation (2.1). In Chapter 3, I will discuss a neural network pipeline modeling $P(\text{score}) \cdot P(\text{performance} \mid \text{score})$ which pairs with a known deterministic synthesizer represent-

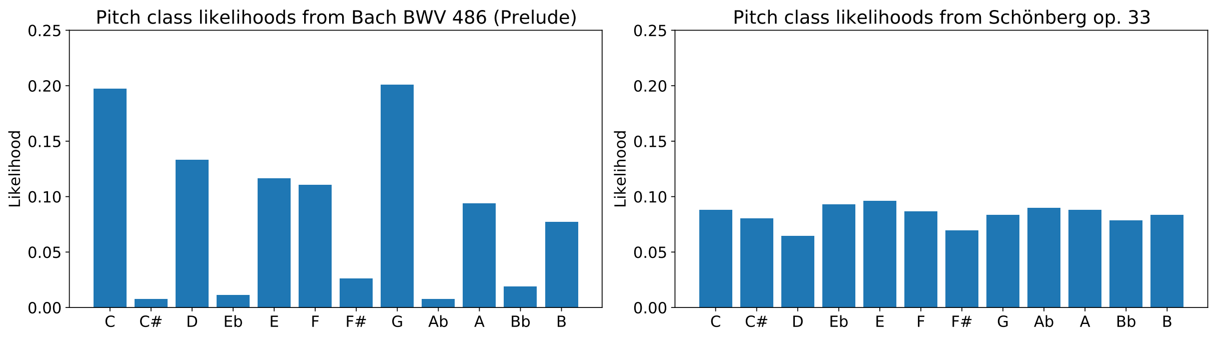


Figure 2.2: Comparing the pitch class frequencies for specific pieces composed by Bach and Schönberg.

ing $P(\text{acoustic} \mid \text{performance})$. In Chapter 4, I will discuss attempts at modeling $P(\text{music})$ directly by training neural networks to generate waveforms. First, I will present a pedagogical oversimplification of modeling $P(\text{score})$ to show how we can intuitively connect probabilistic tooling to our understanding of music.

2.3 Simplifying and generating $P(\text{score})$

To give us a simple playground for relating probabilistic models to our musical knowledge, let's strip away an extreme amount of information from musical scores. Namely, we will map a musical score into a time-ordered list of *equal-tempered pitch classes* used in that score, discarding all rhythmic and octave information.[1] Hence, "Twinkle Twinkle Little Star" would map to `CCGGAAGFFEEDDC` in our simple music representation.

Let's now construct a simple model of $P(\text{score})$ given our chosen representation. The *unigram* distribution is a *statistic*—implying that it is gathered directly from the data rather than learned—which models the frequencies of various "tokens" in a language. Here the unigram distribution amounts to a vector of 12 numbers which represent the proportions with which the pitch classes appear in the data. In Figure 2.2, we compare the unigram distributions for two very

[1]Note that we have already made several assumptions here and restricted our modeling to music which is composed with equal temperament.

different pieces of equal-tempered piano music under our simple music representation: *Prelude in C Major (BWV 846), Bach* (Figure 2.1) and *Klavierstücke, Op. 33, Schönberg*. Even this simplest of musical probability distributions can highlight stylistic distinctions: the unigram distribution of the Bach piece is highly peaked at the 7 pitches of the C major diatonic scale, while that of the Schönberg piece is mostly flat, reflective of his twelve-tone technique.

To *generate* musical sequences of N notes from a pitch class unigram distribution, we could simply sample from the distribution N times. Hence, from our Bach distribution, we would expect around 20 occurrences of the pitch class `G` for $N = 100$. Obviously, this would be a terrible method for music generation, as our probabilistic model preserves none of the sequential aspects of music. However, in addition to generation, a probabilistic model of music offers us the ability to ask questions about the *likelihood* of particular pieces under that model, which often can be useful even if a model represents a poor method for generating music.

Let's examine the likelihood of Twinkle Twinkle Little Star under both of these distributions. We can do this by factorizing $P(score)$ into the product of likelihoods of its constituent pitch classes, where p_n is the pitch class of the nth note

$$P(\text{score}) = P(p_1) \cdot P(p_2) \cdot \ldots \cdot (p_N). \tag{2.2}$$

Because these cascading multiplications quickly result in small values, we usually examine *log-likelihoods*, which vary monotonically with likelihoods and can hence be used for comparison

$$\log P(\text{score}) = \log P(p_1) + \log P(p_2) + \ldots + \log(p_N). \tag{2.3}$$

Under our Bach and Schönberg distributions, the log-likelihoods of Twinkle Twinkle Little Star are -27.1 and -34.7 respectively. Hence, we say that Twinkle Twinkle Little Star is *more likely* under our Bach distribution because $-27.1 > -34.7$. Intuitively, this means that if we were to generate music from this distribution by the process described above, we would be more likely to arrive at Twinkle Twinkle Little Star by sampling from the Bach distribution than from the Schönberg distribution.

It is important to note that one reason Twinkle Twinkle Little Star was more likely under our Bach distribution is because it is in the same key as the Bach Prelude we computed our distribution from. If we instead asked the question about the same song transposed to `Eb` major, the log-likelihoods under our Bach and Schönberg distributions are -46.0 and -34.3 respectively. Hence, the `Eb` major version of the piece is more likely under the Schönberg distribution. Upon closer inspection, we see that the likelihoods assigned by the Schönberg distribution to both transpositions of the piece were similar: -34.7 for `C` major and -34.3 for `Eb` major. This is because the unigram distribution of the Schönberg piece is close to a *uniform* distribution over the 12 pitch classes, where each pitch class would be equally likely ($p = \frac{1}{12}$). Under a perfectly uniform distribution, all transpositions of Twinkle Twinkle Little Star ($N = 14$) would be assigned the same log-likelihood: $14 \log(\frac{1}{12}) = -34.8$.

2.3.1 Preserving sequential aspects of music in $P(\text{score})$

So far we have only examined distributions which discard all of the temporal context in musical sequences. These distributions are somewhat useful for comparing the likelihoods of different pieces but not useful for music generation. Using our same simple music representation, we now turn to distributions which modify their predictions of subsequent pitch classes given knowledge of previous ones. Perhaps the simplest such model is a *bigram* distribution, often referred to as a Markov chain. In this case, a bigram distribution describes the frequencies of transitions from one pitch class to the next

$$P(\text{score}) = P(p_1) \cdot P(p_2 \mid p_1) \cdot \ldots \cdot P(p_N \mid p_{N-1}). \tag{2.4}$$

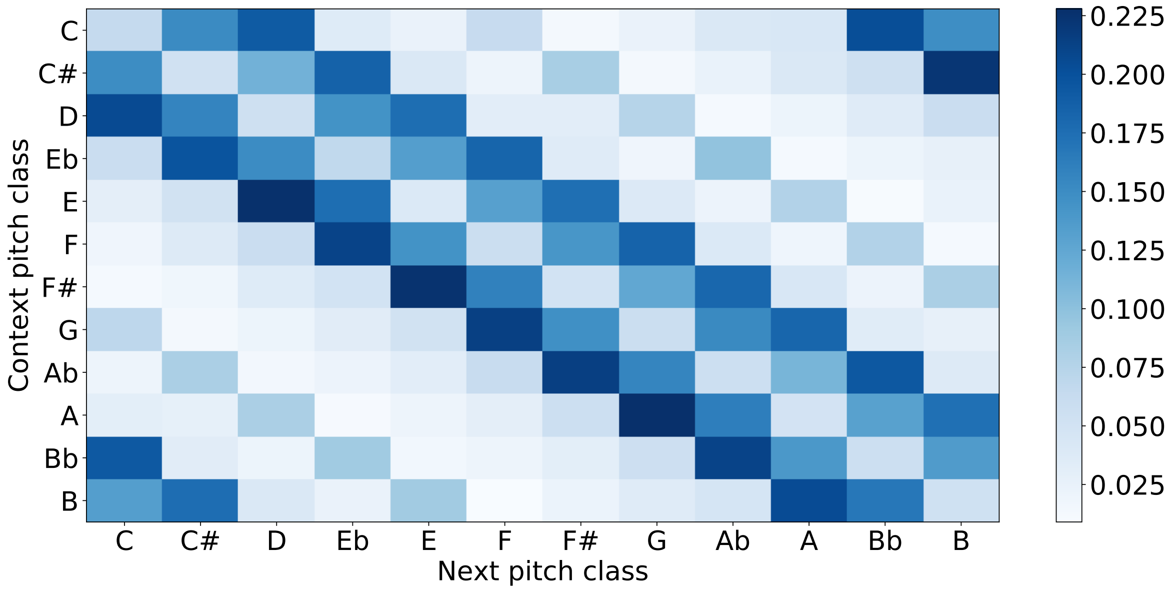


Figure 2.3: Pitch class transition likelihoods across all of Bach's *Well-Tempered Clavier* (BWV 846–893). The transitions reflect our understanding of Bach's compositional style. For example, major seconds are the most likely intervals while tritones are the least likely.

Hence, it is a matrix with 12 rows and columns where cell i, j is the likelihood that pitch class j should come next given that the previous pitch class was i.

To create such a distribution, we gather statistics from our corpus about the relative proportions of such transitions, similarly to how we gathered the statistics for the pitch class likelihoods for the unigram distribution. Rather than just the first prelude, let's now examine the entirety of J.S. Bach's *Well-Tempered Clavier* (BWV 846–893), which contains pieces in all 12 key signatures. I display a visualization of the bigram distribution for this corpus in Figure 2.3. It is easy to relate certain aspects of our knowledge of Bach's music to this visualization. For example, transitions along the diagonal correspond to using the same pitch class for two notes in a row. As we expect, these transitions are less likely than localized transitions such as moving up or down by a major or minor 2nd. There is noticeably low likelihood assigned to tritone intervals, which also agrees with our knowledge of Bach's compositional style.

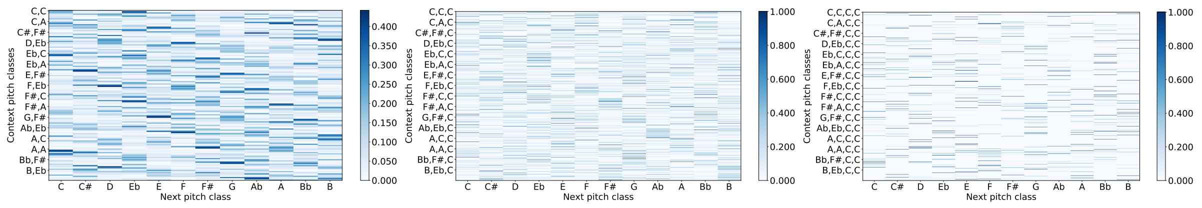


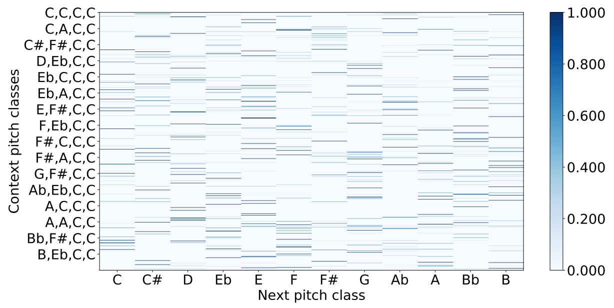
Figure 2.4: Trigram, 4-gram, and 5-gram models for Bach's *Well-Tempered Clavier*. While the exponential growth of the model parameters make it difficult to interpret the distributions visually, we can see that the sparsity of the models (how many n-grams were *not* present in the corpus) increases as the context grows (i.e., there is more whitespace in the diagram).

We can further extend our context by adopting a trigram model:

$$P(\text{score}) = P(p_1) \cdot P(p_2 \mid p_1) \cdot \ldots \cdot P(p_N \mid p_{N-2}, p_{N-1}), \tag{2.5}$$

Using one and two more tokens of context, we arrive at 4- and 5-gram distributions respectively. In Figure 2.4, we show a visualization of these distributions computed on the same corpus as our bigram distribution. They are difficult to interpret visually, as the number of model parameters increases exponentially with linear growth of the context. However, they do illustrate an important limitation of n-gram models: that the proportion of possible sequences of length n that occur in our dataset becomes smaller (more white space in the visualization) as we increase the size of n.

When we calculated our bigram distribution, all of the 144 possible two-pitch bigrams occurred at least once in the training data. For our 5-gram distribution however, only 8% of the possible 5-grams ever occur. We are now faced with an issue when we want to assess the likelihood for one of the unobserved 5-grams: **is this sequence unlikely because Bach would have never written it, or because he never got around to writing it?** This illustrates one reason why n-gram distributions are insufficient for modeling music. The more we increase n (and therefore increase our capacity for representing long-term structure), the less information we can extract from our dataset.



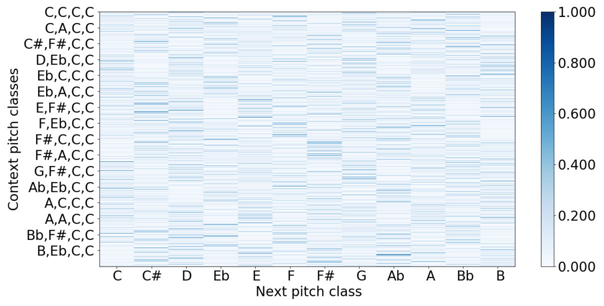


Figure 2.5: Comparing likelihoods assigned to sequences of five pitch classes by a 5-gram distribution (left) and a recurrent neural network (right) trained on the same corpus. The recurrent neural network has learned a smooth parameterization of the distribution that allows us to estimate reasonable likelihoods for sequences which did not appear in the training data.

2.3.2 Language models for $P(\text{score})$

In Chapter 3, I discuss the usage of *language models* for modeling $P(\text{score})$ (in the context of representations more complex than the simplified one we examine here). A language model generalizes the n-gram factorization to use *all* of the previous context in the sequence:

$$P(score) = P(p_1) \cdot P(p_2) \cdot \ldots \cdot P(p_N \mid p_1, \ldots, p_{N-1}). \tag{2.6}$$

Due to the sparsity issue described in the previous section, we would be unable to extract an effective model for this factorization by simply counting the occurrences in our training data as we did for our n-gram models. Instead, we will use neural networks to parameterize a smooth version of this distribution, so that we can learn about the likelihood of transitions we have never seen before. One common way of achieving this is to use recurrent neural networks, which learn to summarize context up to step N into a continuous vector $\boldsymbol{h}_N$ using parameters θ. Hence, we are modeling the following proxy to Equation (2.6):

$$P(score) = \prod_{i=1}^{N} P_\theta(p_N \mid p_{N-1}, \boldsymbol{h}_{N-1}). \tag{2.7}$$

This constructions makes it possible for the same neural network to make predictions for a different amounts of past context. In Figure 2.5, we see that this produces predictions for 5-grams that are much less sparse.

You can hear “music” generated by the *n*-gram and RNN pitch class models described in this section in my online supplementary material https://bit.ly/2L1IxYZ .

Chapter 3

Music generation in the symbolic domain

In this chapter, we extend recent results for symbolic piano music generation [Huang et al., 2019] to the multi-instrumental setting. Both piano and multi-instrumental music are *polyphonic*, where multiple notes may be sounding at any given point in time. However, the generation of multi-instrumental music presents an additional challenge not present in the piano domain: handling the intricate interdependencies between multiple instruments. Another obstacle for the multi-instrumental setting is that there is less data available than for piano, making it more difficult to train the types of powerful generative models used in [Huang et al., 2019].

Until recently, music generation methods struggled to capture two rudimentary elements of musical form: long-term structure and repetition. Huang et al. [Huang et al., 2019] demonstrated that powerful neural network *language models*, i.e., models which assign likelihoods to sequences of discrete tokens, could be used to model—and subsequently generate—classical piano music with long-term structure. To our ears, the piano music generated from their *Music Transformer* model represents the most compelling computer-generated music to date. In order to adapt this method to the multi-instrumental setting we incorporate the semantics of the instruments directly into our language-like music representation. However, this strategy alone may be insufficient to generate high-quality multi-instrumental music, as the results of [Huang et al., 2019] also depend on access to large quantities of piano music.

To begin to address the data availability problem, we focus on an unusually large dataset of multi-instrumental music. The *Nintendo Entertainment System Music Database* (NES-MDB) [Donahue et al., 2018c] contains 46 hours of *chiptunes*, music written for the four-instrument ensemble of the NES (video game system) sound chip. This dataset is appealing for music generation research not only for its size but also for its structural homogeneity—all of the music is written for a fixed ensemble. It is, however, smaller than the 172 hours of piano music in the *MAESTRO Dataset* [Hawthorne et al., 2019] used to train Music Transformer.

The largest available source of symbolic music data is the *Lakh MIDI Dataset* [Raffel, 2016] which contains over 9000 hours of music. This dataset is structurally heterogeneous (different instruments per piece) making it challenging to model directly. However, intuition suggests that we might be able to benefit from the musical knowledge ingrained in this dataset to improve our performance on chiptune generation. Accordingly, we propose a procedure to heuristically map the arbitrary ensembles of music in Lakh MIDI into the four-voice ensemble of the NES. We then pre-train our generative model on this dataset, and fine-tune it on NES-MDB. We find that this strategy improves the quantitative performance of our generative model by 10%. Such transfer learning approaches are common practice in state-of-the-art natural language processing [Devlin et al., 2018, Radford et al., 2019], and here we develop new methodology to employ these techniques in the music setting.

We refer to the generative model pre-trained on Lakh MIDI and fine-tuned on NES-MDB as *LakhNES*. In addition to strong quantitative performance, we also conduct multiple user studies indicating that LakhNES produces strong qualitative results. LakhNES is capable of generating chiptunes from scratch, continuing human-composed material, and producing melodic material corresponding to human-specified rhythms. Sound examples can be found here: https://bit.ly/2PtQTXK .

3.1 Related work

Music generation has been an active area of research for decades. Most early work involved manually encoding musical rules into generative systems or rearranging fragments of human-composed music; see [Nierhaus, 2009] for an extensive overview. Recent research has favored machine learning systems which automatically extract patterns from corpora of human-composed music.

Many early machine learning-based systems focused on modeling simple *monophonic* melodies, i.e., music where only one note can be sounding at any given point in time [Todd, 1989, Mozer, 1994, Eck and Schmidhuber, 2002]. More recently, research has focused on *polyphonic* generation tasks. Here, most work represents polyphonic music as a *piano roll*—a sparse binary matrix of time and pitch—and seeks to generate sequences of individual piano roll timesteps [Boulanger-Lewandowski et al., 2012, Johnson, 2017, Donahue et al., 2018c] or chunks of timesteps [Yang et al., 2017]. Other work favors an *event-based* representation of music, where the music is flattened into a list of musically-salient events [Simon and Oore, 2017, Mao et al., 2018, Huang et al., 2019]. None of these methods allow for the generation of multi-instrumental music, as they provide no mechanism for mapping the generated polyphonic scores to individual instruments.

Other research focuses on the multi-instrumental setting and seeks to provide systems which can *harmonize* with human-composed material [Allan and Williams, 2005, Huang et al., 2017, Hadjeres and Pachet, 2017, Yan et al., 2018]. Unlike the system we develop here, these approaches all require complex inference procedures to generate music without human input. A recent paper [Dong and Yang, 2018] attempts multi-instrumental music generation from scratch, but their method is limited to generating fixed lengths, unlike our method which can generate arbitrarily-long sequences. There is also an increasing amount of music generation research that focuses on generating music in the audio domain [Andreux and Mallat, 2018, Donahue et al., 2019a, Dieleman et al., 2018], though this work is largely unrelated to symbolic domain methods.

3.2 Dataset preliminaries

Our *NES Music Database* (NES-MDB) [Donahue et al., 2018c] consists of approximately 46 hours of music composed for the sound chip on the Nintendo Entertainment System. This dataset is enticing for research in multi-instrumental music generation because (1) it is an unusually large corpus of music that was composed for a fixed ensemble, and (2) it is available in symbolic format.

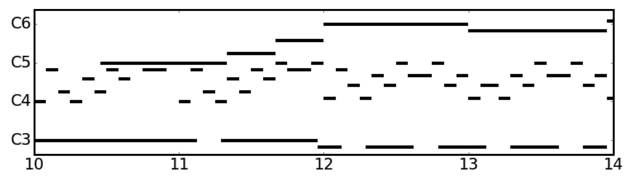
3.2.1 NES ensemble preliminaries

The ensemble on the NES sound chip has four monophonic instrument voices: two pulse waveform generators (P1/P2), one triangle waveform generator (TR), and one noise generator (NO).[1] The first three of these instruments are melodic voices: typically, TR plays the bass line and P1/P2 are interchangeably the melody and harmony. The noise instrument is used to provide percussion.

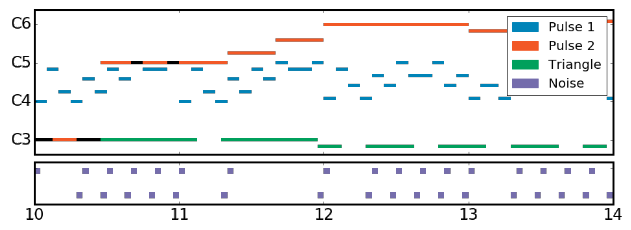
The various instruments have a mixture of sound-producing capabilities. For example, the range of MIDI pitches which P1/P2 can generate is 33–108, while the range of TR extends an octave lower (21–108). The noise channel can produce 16 different "types" of noise which correspond to different center frequencies and bandwidths. Each instrument also has a variety of dynamics and timbral attributes. It is shown in [Donahue et al., 2018c] that these *expressive* attributes can be estimated from the score post-hoc, and hence we ignore them in this study to focus on the problem of modeling composition rather than expressive performance.

Each chiptune in NES-MDB is stored as a MIDI file, and the constituent MIDI events are quantized at audio rate (44100 *ticks* per second). Paired with code which synthesizes these MIDI

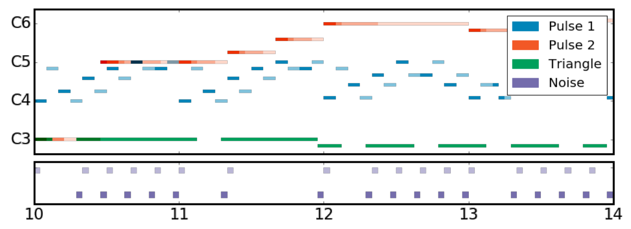
[1]There is an additional fifth voice capable of waveform playback that the authors of NES-MDB excluded.



(a) Blended score (degenerate)



(b) Separated score (melodic voices **top**, percussive voice **bottom**)



(c) Expressive score (includes dynamics and timbral changes)

Figure 3.1: Four representations for a segment of *Ending Theme* from *Abadox* (1989) by composer Kiyohiro Sada. The blended score (fig. 3.1a), used in prior polyphonic generation research, is degenerate when multiple voices play the same note.

files as NES audio, the files contain all of the information needed to synthesize the original 8-bit waveforms.

3.3 Representations and tasks

Using the terminology of Section 2.2, here we are interested in modeling $P(\text{score})$, and subsequently modeling $P(\text{performance} \mid \text{score})$ as a pipeline for generating *expressive* chiptunes. Samples from the latter are fed into a deterministic synthesizer, (instead of attempting to *model* $P(\text{acoustic} \mid \text{performance})$, resulting in samples from a portion of $P(\text{music})$ which are recognizable as chiptunes.

We design a number of different representations which beget several different tasks. I describe them all in this section for thoroughness, but I note that **the remainder of this chapter will only focus on two tasks: event-based score generation (Section 3.3.3) and expressive performance generation (Section 3.3.4)**.

3.3.1 Blended score representation and task

Much of the prior research on polyphonic music generation [Boulanger-Lewandowski et al., 2012, Chung et al., 2014, Johnson, 2017] represents music in a blended score representation. A blended score B is a sparse binary matrix of size $N \times T$, where N is the number of possible note values, and $B[n,t] = 1$ if any voice is playing note n at timestep t or 0 otherwise (fig. 3.1a). Often, N is constrained to the 88 keys on a piano keyboard, and T is determined by some subdivision of the meter, such as sixteenth notes. When a polpyhonic score $\boldsymbol{s}$ is represented by B, statistical models often factorize the distribution as a language model of *chords*, the columns B_t:

$$P(\boldsymbol{s}) = P(B_1) \cdot P(B_2 \mid B_1) \cdot \ldots \cdot P(B_T \mid B_{t<T}). \tag{3.1}$$

This representation simplifies the overall task, but it is problematic for music with multiple instruments (such as the music in NES-MDB). Resultant systems must provide an additional mechanism for assigning notes of a blended score to instrument voices, or otherwise render the music on polyphonic instruments such as the piano. Therefore, we mainly explore this task to compare our methods to prior work.

3.3.2 Separated score representation and task

Given the shortcomings of the blended score, we might prefer models which operate on a separated score representation (fig. 3.1b). A separated score S is a matrix of size $V \times T$, where V is the number of instrument voices, and $S[v,t] = n$, the note n played by voice v at timestep t. In other words, the format encodes a monophonic sequence for each instrument voice.

Statistical approaches to this representation can explicitly model the relationships between various instrument voices by

$$P(\boldsymbol{s}) = \prod_{t=1}^{T} \prod_{v=1}^{V} P(S_{v,t} \mid S_{v,\hat{t} \neq t}, S_{\hat{v} \neq v, \forall \hat{t}}). \tag{3.2}$$

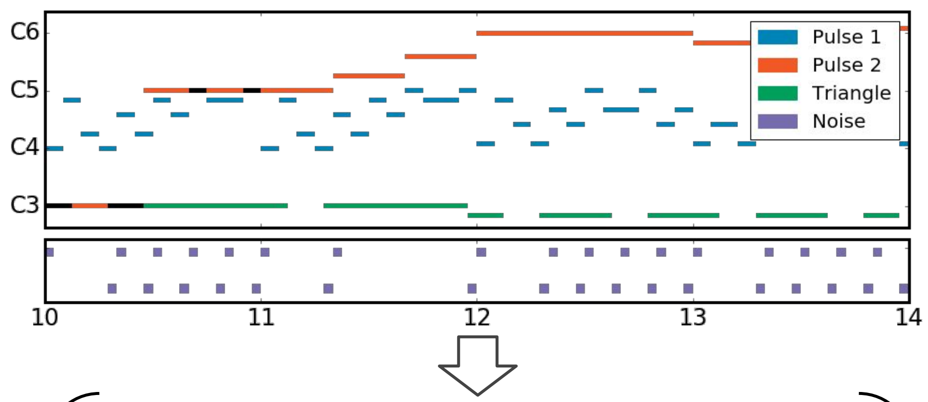
This formulation explicitly models the dependencies between $S_{v,t}$, voice v at time t, and every other note in the score. For this reason, Equation (3.2) more closely resembles the process by which human composers write multi-instrumental music, incorporating temporal and contrapuntal information. Another benefit is that resultant models can be used to harmonize with existing musical material, adding voices conditioned on existing ones. However, any non-trivial amount of temporal context introduces high-dimensional interdependencies, meaning that such a formulation would be challenging to sample from. As a consequence, solutions are often restricted to only take past temporal context into account, allowing for simple and efficient ancestral sampling (though Gibbs sampling can also be used to sample from eq. (3.2) [Hadjeres and Pachet, 2017, Huang et al., 2017]).

Most existing datasets of multi-instrumental music (the chorales of J.S. Bach being a notable exception [Allan and Williams, 2005]) have uninhibited polyphony, causing a separated score representation to be inappropriate. However, the hardware constraints of the NES APU impose a strict limit on the number of voices, making the format ideal for NES-MDB.

3.3.3 Event-based representation and task

Because no tempo or beat information exists in NES-MDB, for our piano roll representations we are forced to discretize the time axis to a fixed rate of 24 timesteps per second. This high rate is necessary for capturing nuanced timing information in the scores but results in much of the information being redundant across adjacent timesteps. This represents a challenge as long-term dependencies are a barrier to success for sequence modeling with machine learning.

To circumvent these issues, we design an *event-based* representation (bottom of Figure 3.2)



NO_NOTEON_13, DT_732, NO_NOTEOFF, DT_2197,
P1_NOTEON_76, DT_2, P1_NOTEON_70, DT_1463,
P2_NOTEON_87, DT_1, P2_NOTEOFF, DT_148,
...
P1_NOTEON_61, DT_41, P2_NOTEON_85, DT_26,
TR_NOTEON_39, DT_2, TR_NOTEON_43, DT_23,
NO_NOTEON_9, DT_646, NO_NOTEON_13, DT_733

Figure 3.2: A visual comparison between the piano roll representation of the original NES-MDB paper [Donahue et al., 2018c] (**top**) and the event representation of this work (**bottom**). In the piano roll representation, the majority of information is the same across timesteps. In our event representation, each timestep encodes a musically-meaningful change.

similar to that used for single-instument music in [Simon and Oore, 2017]. Specifically, we convert each NES-MDB MIDI file into a time-ordered sequence of events, so that every entry in the sequence corresponds to a musically-salient occurrence.

To handle the rhythmic information, we add time shift (ΔT) events which represent time advancing by some discrete number of ticks (each tick is $\frac{1}{44100}$th of a second). To keep the number of events in our representation tractable, we quantize ΔT events in the real data to fixed gratings. We embed the multi-instrumental aspect of our problem directly into this representation by using separate note on/off events for each instrument. Our final representation consists of 631 events, of which about half encode time-related events and half note-related (Table 3.1). Apart from minor timing quantization errors, this format is a lossless transformation of the original MIDI score.

Table 3.1: Schematic for our event-based representation of NES-MDB, reminiscent of the one used in *Performance RNN* [Simon and Oore, 2017]. The 631 events in our representation are distributed among time-shift (ΔT) events (which allow for nuanced timing), and note off/on events for individual instruments (as in typical MIDI).

Event description	Event ID(s)
Start or end of sequence	0
ΔT for 1–100 ticks (short)	1–100
ΔT for 100–1000 ticks (medium)	101–190
ΔT for $>$ 10000 ticks (long)	191–370
P1 Note Off/On	371–447
P2 Note Off/On	448–524
TR Note Off/On	525–613
NO Note Off/On	614–630

3.3.4 Expressive performance generation

Given a piece of a music, a skilled performer will embellish the piece with *expressive characteristics*, altering the timing and dynamics to deliver a compelling rendition. While a few instruments have been augmented to capture this type of information symbolically (e.g. a Disklavier), it is rarely available for examination in datasets of multi-instrumental music. Because NES music is comprised of instructions that recreate an exact rendition of each piece, expressive characteristics controlling the velocity and timbre of each voice are available in NES-MDB. Thus, each piece can be represented as an *expressive score* (fig. 3.1c), the union of its separated score and expressive characteristics.

We consider the task of mapping a separated score (Figure 3.1b) $\boldsymbol{s}$ onto expressive characteristics $\boldsymbol{e}$. Hence, we would like to model $P(\boldsymbol{e} \mid \boldsymbol{c})$. This allows for a convenient pipeline for music generation where a piece of music is first composed with binary amplitudes and then mapped to realistic dynamics, as if interpreted by a performer.

3.4 Methodology

Here we describe our primary methodology which focuses on the event-based task Section 3.3.3. To model the event sequences outlined in the last section, we adopt a *language modeling* factorization. We factorize the joint probability of a musical sequence consisting of N events $(E_1, \ldots, E_N)$ into a product of conditionals:

$$P(E_1) \cdot P(E_2 \mid E_1) \cdot \ldots \cdot P(E_N \mid E_1, \ldots, E_{N-1}). \tag{3.3}$$

This factorization is convenient because it allows for a simple left-to-right algorithm for generating music: sampling from the distribution estimated by the model at each timestep (conditioned on previous outputs). The goal of our optimization procedure is to find a model configuration which maximizes the likelihood of the real event sequences. Motivated by the strong results for piano music generation from the recent *Music Transformer* [Huang et al., 2019] approach, we also adopt a Transformer [Vaswani et al., 2017] architecture.

3.4.1 Transformer architecture

The Transformer [Vaswani et al., 2017] is an *attention*-based neural network architecture. In our context, this means that the model has a mechanism which explicitly biases its predictions based on a subset of musical events that have happened in the past. The model's design gives it the ability to *learn* which subset of past musical events to pay attention to when predicting the current event. This mechanism may be especially useful for learning patterns of repetition in music across large gaps of time.

The original Transformer architecture [Vaswani et al., 2017] was an encoder-decoder model designed for language translation. In this paper, we are only concerned with the decoder portion of the Transformer. Our work uses a recent extension of Transformer called Transformer-XL [Dai et al., 2019], which is designed specifically to handle longer sequences. Transformer-XL builds upon the Transformer architecture by augmenting it with a recurrence mechanism. The recurrence mechanism enables Transformer-XL to use information beyond its training segment by learning how to incorporate recurrent state from previous segments. In contrast, the original Transformer is only able to alter its predictions based on the current training segment, hence the available system memory during training is a bottleneck to its ability to learn long-term dependencies. In order to effectively use its recurrent state, Transformer-XL adopts a sophisticated position-aware mechanism so the model can generalize to different amounts of recurrent memory during generation.

The Music Transformer [Huang et al., 2019] is a different Transformer variant that also attempts to tackle long-range dependencies by using a mechanism which reduces the quadratic memory cost of attention, enabling training on longer sequences. Although similar in goal to Transformer-XL, its method is orthogonal and could, in theory, be combined with the recurrent mechanism of Transformer-XL. For simplicity, we focus on the Transformer-XL architecture as its recurrence mechanism alone is sufficient to learn long-term dependencies. Additionally, code to reproduce the Music Transformer method is unavailable.

3.4.2 Pre-training

Transformers are extremely high-dimensional models, and accordingly they can learn effective strategies for extremely large datasets [Radford et al., 2019]. One barrier to their application in the music domain is that most symbolic music datasets are either too small or too structurally heterogeneous. For example, the popular Bach chorales dataset [Hild et al., 1992] is structurally homogeneous (all chorales have four voices), but small (only 306 chorales). In contrast, the Lakh MIDI dataset [Raffel, 2016] is enormous (175k songs) but heterogeneous (varying numbers of instruments per piece). The NES-MDB dataset we use in this work represents a middle ground (large and structurally-homogenous), but is still substantially smaller than the MAESTRO dataset [Hawthorne et al., 2019] used to train Music Transformer (46 hours vs. 172

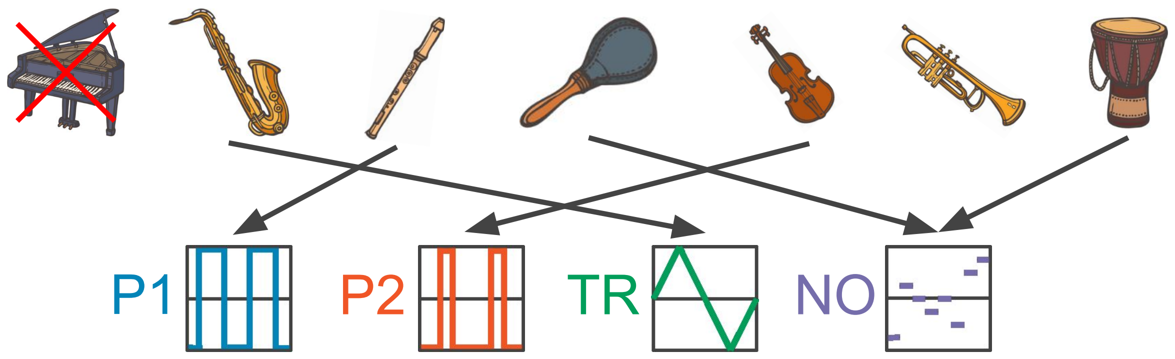


Figure 3.3: Illustration of our mapping heuristic used to enable transfer learning from Lakh MIDI to NES-MDB. We identify monophonic instruments from the arbitrary ensembles in Lakh MIDI and randomly assign them to the fixed four-instrument ensemble of NES-MDB.

hours).

We hypothesize that we can improve the performance of our model on our NES music generation task by leveraging the musical information in the larger Lakh MIDI dataset. To test this, we propose a two-step procedure. First, we map each structurally-heterogeneous Lakh MIDI file into one which can be performed by our NES ensemble. Then, we pre-train a Transformer on this dataset, and fine-tune this pre-trained model on the NES-MDB dataset. Such transfer learning procedures are common methodology in other areas of machine learning [Pan and Yang, 2010], but remain hitherto unexplored in music generation research. One possible reason for the lack of investigation into this strategy is that mapping music from one domain to another requires careful consideration of musical invariants, and hence is less straightforward than analogous methodology for other tasks (e.g. language). We consider this transfer learning protocol to be a primary methodological contribution of this work.

Mapping Lakh MIDI to the NES ensemble

Here we describe our protocol for mapping Lakh MIDI data into a score suitable for the four monophonic instruments of the NES ensemble. For a given example from Lakh MIDI, we first identify all of its monophonic melodic instruments (skipping the example if it has no such instruments). Then, we filter out instruments which fall outside of the range of MIDI notes that the NES ensemble is capable of producing (Section 3.2.1). We randomly assign these instruments to the three melodic instruments of the NES (P1/P2/TR) (Figure 3.3). Because there are a variable number of instruments in each Lakh MIDI example, there are potentially many possible assignments. Hence, we output multiple examples for each input Lakh MIDI example, up to a maximum of 16 per input.

In addition to this strategy for melodic instruments, we also design a strategy for mapping percussive instruments in Lakh into the percussive noise instrument of the NES ensemble. We first identify percussive instruments in each Lakh MIDI example. Then, each individual percussive voice (e.g. snare drum, hi-hat) is randomly assigned to a noise "type" (1–16), emulating how the noise instrument is used by human composers to encode syncopated rhythms.

From the 175k MIDI files in Lakh MIDI, our mapping procedure produces 775k examples suitable for performance by the NES ensemble. It is straightforward to imagine similar mapping procedures for other ensembles (e.g. string quartet, vocal choir), and thus it is possible that music generation research in other domains could reuse this procedure to enable transfer learning.

3.5 Event-based score generation experiments

We first conduct an experiment to train Transformer-XL [Dai et al., 2019] on our event representation (Section 3.3.3) of NES-MDB. We train the model on excerpts from the training data of 512 events; each excerpt represents around 9 seconds of music on average. Because of the recurrent attention mechanism in Transformer-XL, the model effectively has access to twice this length in its history.

We use the smaller configuration of Transformer-XL which has 12 attention layers each with 8 heads (full details of our model can be found in our code which will be released upon publication). The learning rate 2e−4 used to train this model on text was found to be too high for our musical application, so we lowered it to 2e−5. Training was stopped when the performance of the model on the validation data stopped improving. We trained the model using four NVIDIA Titan X GPUs with minibatches of size 30, and it reached its early stopping criteria in less than a day. Pre-trained models will be released upon publication.

3.5.1 Data augmentation and pre-training

To improve the performance of our model further, we employed standard music data augmentation methods as well as ones which we developed specifically for the multi-instrumental setting:

1. (Standard) Transpose melodic voices by a random number of semitones between -6 and 5 (inclusive).
2. (Standard) Adjust the speed of the piece by a random percentage between $\pm 5\%$.
3. Half of the time, remove a random number of instruments from the ensemble (leaving at least one).
4. Half of the time, shuffle the score-to-instrument alignment for the melodic instruments only (e.g. TR performs P2's part).

Finally, we experimented with pre-training our model on the Lakh MIDI dataset mapped to the NES ensemble (Section 3.4.2). To conduct this experiment, we first split the Lakh data into training and validation subsets. We then trained the model for a week on the training set (with data augmentation) and monitored performance on the validation set. Because of the extreme size of the dataset, the model only completed four epochs of training. Even after a week, the model was underfitting the training data (validation performance was still improving). We then fine-tuned this pre-trained model on the NES-MDB training data, again performing early stopping based on the validation performance. Both our pre-training and fine-tuning experiments use the same hyperparameters outlined in the previous section.

3.5.2 Baselines

We also measure the performance of competitive baselines on our event-based representation of NES-MDB. Our simplest baselines consist of *n*-gram models, i.e., statistics gathered directly from the training data of how often certain length-*n* sequences appear. Specifically, we build unigram (1-gram) and 5-gram models, using backoff for the latter to provide a likelihood for 5-grams which are not present in the training data. We also compare to an LSTM [Hochreiter and Schmidhuber, 1997] recurrent neural network, which is a popular model for music generation. Our LSTM is configured so that it has approximately the same number of parameters as our Transformer-XL model.

3.5.3 Quantitative analysis

Table 3.2: Quantitative performance of various models trained on the event-based representation (631 event types) of NES-MDB. *Params* indicates the number of parameters of each model. *Epochs* is the number of data epochs the model observed before early stopping based on the validation data. *Test PPL* represents the perplexity of the model on the test data, i.e., the exponentiation of its average negative log-likelihood on the test data. A *lower* perplexity indicates that the model *better* fits this unseen data.

Model	Params	Epochs	Test PPL
Random	0	0	631.00
Unigram	631	1	198.14
5-gram	9M	1	37.25
LSTM [Hochreiter and Schmidhuber, 1997]	40M	18	14.11
+Data augmentation		35	12.64
Transformer-XL [Dai et al., 2019]	41M	76	3.50
+Data augmentation		350	2.74
+Pre-train (LakhNES)		250	**2.46**

We report the *perplexity* (PPL) of each model on the test set in Table 3.2. Perplexity is calculated by first averaging the negative log-likelihood of each model across the test data, then exponentiating the average, i.e., $e^{\frac{1}{N}\sum_{i=1}^{N} -\log q_i}$, where q_i is the likelihood assigned by a given model to the *i*-th event. A lower perplexity on the test set indicates that a model is a good fit for unseen data, and hence increases our confidence in its ability to generate new music.

We find that Transformer-XL dramatically outperforms both the *n*-gram and LSTM baselines on the NES-MDB event-based task (PPL of 3.5 vs. 37.2 and 14.1 respectively). Data augmentation improves the performance of both the LSTM and Transformer-XL (by 10% and 22% respectively), and also increases the number of epochs the models observe before overfitting. We observe that LakhNES (Transformer-XL pre-trained on Lakh MIDI and fine-tuned on NES-MDB with augmentation), achieves 10% better performance than training with data augmentation alone.

We also conduct an experiment to measure the performance effect of using different amounts of Lakh MIDI pre-training before fine-tuning on NES-MDB. Specifically, we measure the performance on the NES-MDB fine-tuning task after 1, 2, and 4 epochs of Lakh MIDI pre-training. We plot the test PPL of each model after fine-tuning in Figure 3.5. The results agree with our expectation that increasing the amount of pre-training improves the fine-tuned model's performance, though with diminishing returns.

3.6 Expressive performance generation experiments

The expressive performance task consists of learning a mapping from a separated score to suitable expressive characteristics. Each timestep of a separated score in NES-MDB has note information (random variable N) for the four instrument voices. An expressive score additionally has velocity (V) and timbre (T) information for P1, P2, and NO but not TR. We can express the distribution of performance characteristics given the composition as $P(V,\ T \mid N)$. Some of our proposed solutions factorize this further into a conditional autoregressive formulation $\prod_{t=1}^{T} P(V_t, T_t \mid N, V_{\hat{t}<t}, T_{\hat{t}<t})$, where the model has explicit knowledge of its decisions for velocity and timbre at earlier timesteps.

Unlike for separated composition, there are no well-established baselines for multi-

Table 3.3: Results for expressive performance experiments evaluated at points of interest (POI). Results are broken down by expression category (e.g. V_{NO} is noise velocity, T_{P1} is pulse 1 timbre) and aggregated at POIs and globally (All).

	Negative log-likelihood						
	Single voice					Aggregate	
Model	V_{P1}	V_{P2}	V_{NO}	T_{P1}	T_{P2}	POI	All
Random	2.77	2.77	2.77	1.39	1.39	11.09	11.09
Unigram	2.87	2.89	3.04	1.35	1.33	11.47	9.65
Bigram	2.82	2.85	2.78	4.27	4.27	17.00	4.57
MultiReg Note	2.74	2.72	2.23	1.27	1.18	10.13	8.49
MultiReg Note+Auto	2.58	2.56	2.04	2.90	2.48	12.56	4.32
LSTM Note	2.68	2.63	2.09	1.32	1.21	9.94	8.28
LSTM Note+Auto	1.93	1.89	1.99	2.23	1.89	9.93	3.42

instrumental expressive performance, and thus we design several approaches. For the autoregressive formulation, our most-sophisticated model (fig. 3.4) uses a bidirectional LSTM to process the separated score, and a forward-directional LSTM for the autoregressive expressive characteristics. The representations from the composition and autoregressive modules are merged and processed by an additional dense layer before projecting to six softmaxes, one for each of V_{P1}, V_{P2}, V_{NO}, T_{P1}, T_{P2}, and T_{NO}. We compare this model (LSTM Note+Auto) to a version which removes the autoregressive module and only sees the separated score (LSTM Note).

We also measure performance of simple multinomial regression baselines. The non-autoregressive baseline (MultiReg Note) maps the concatenation of N_{P1}, N_{P2}, N_{TR}, and N_{NO} directly to the six categorical outputs representing velocity and timbre (no temporal context). An autoregressive version of this model (MultiReg Note+Auto) takes additional inputs consisting of the previous timestep for the six velocity and timbre categories. Additionally, we show results for simple baselines (per-category unigram and bigram distributions) which do not consider N. Because the noise timbre field T_{NO} is so rarely used (less than 0.2% of all timesteps), we exclude it from our quantitative evaluation. Results are shown in table 3.3.

Similarly to the musical notes in the separated composition task, the high rate of NES-

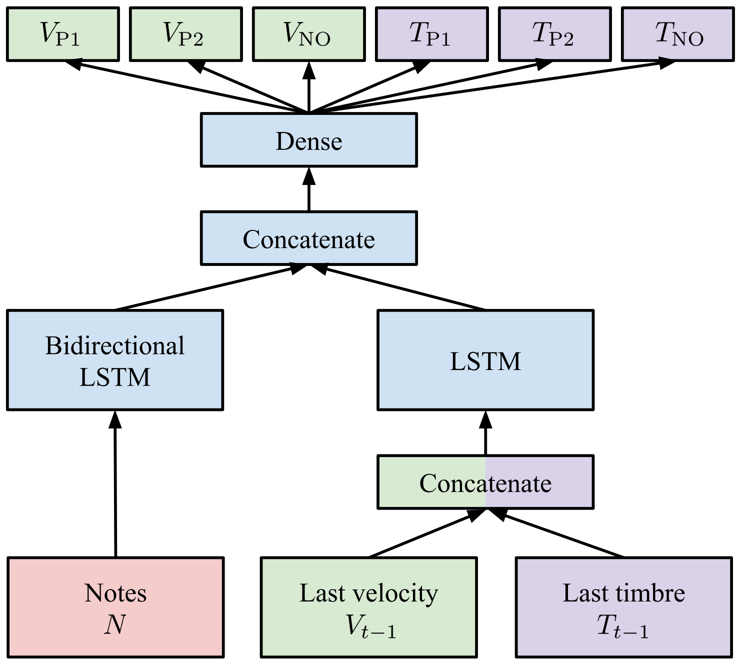


Figure 3.4: LSTM Note+Auto expressive performance model that observes both the score and its prior output.

MDB results in substantial redundancy across timesteps. Averaged across all velocity and timbre categories, any of these categories at a given timestep has a 74% chance of having the same value as the previous timestep.

The performance of the LSTM Note model is comparable to that of the LSTM Note+Auto model at POIs, however the global performance of the LSTM Note+Auto model is substantially better. Intuitively, this suggests that the score is useful for knowing *when* to change, while the past velocity and timbre values are useful for knowing *what* value to output next. Interestingly, the MultiReg Note model has better performance at POIs than the MultiReg Note+Auto model. The latter overfit more quickly which may explain its inferior performance despite the fact that it sees strictly more information than the note-only model.

3.7 User study

Here we perform a user study using our most promising score generation models from Section 3.5. While perplexity is a useful quantitative metric for model comparison, it is not necessarily correlated with human judgements. Since we ultimately seek models which produce music that is convincing to humans, we conduct two user studies on Amazon Mechanical Turk to compare the performance of various models. In both of our user studies we compare four models: (1) 5-gram model, (2) LSTM trained with data augmentation, (3) Transformer-XL trained with data augmentation, and (4) LakhNES (Transformer-XL pre-trained on Lakh MIDI and fine-tuned on NES-MDB). Random examples from all of our methods can be heard at the bottom of our sound examples page: https://bit.ly/2PtQTXK .

3.7.1 Turing test

This study seeks to determine the ability of humans to distinguish between real (human-composed) and fake (computer-generated) chiptunes in a "Turing test" setting. We present human judges with pairs of examples where one example is real and the other fake, and ask them to identify the real example between the two.

We first amass a collection of 5-second audio clips from all of our methods and from the real data. Then, we create pairs of examples where one example is real and the other fake (randomly chosen from our four methods). Given that Mechanical Turk studies are notoriously noisy, we also create control pairs where the fake data comes from a random model (i.e. we generate "music" by selecting events uniformly at random—row 1 in Table 3.2).

We ask human judges to annotate 800 batches each consisting of 10 randomly-ordered pairs, where fake data in 2 of the pairs came from the control set and fake data in 8 of the pairs came from our four methods. For their judgments to be included in our results, workers were required to complete at least 3 batches and achieve 100% accuracy on the 6 control examples in

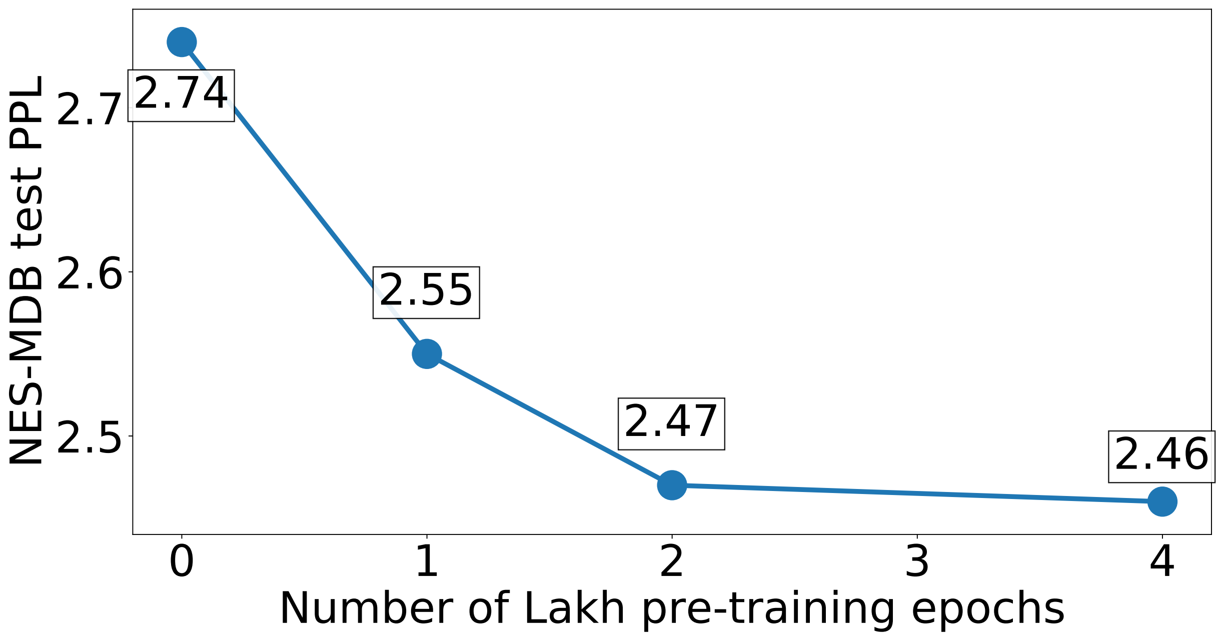


Figure 3.5: Measuring the performance improvement when doubling the amount of Lakh MIDI pre-training before fine-tuning Transformer-XL on NES-MDB. Each datapoint represents the result of a fine-tuning run starting from 0, 1, 2, or 4 epochs of Lakh MIDI pre-training. Additional amounts of pre-training appear to improve performance, though with diminishing returns.

those batches—a worker answering randomly would only be included 1.6% of the time. After filtering, each method was evaluated around 180 times. We report accuracy in Figure 3.6.

In this setting, a lower accuracy indicates that a given model's results sound more human-like, because they were incorrectly identified as human-composed more often. An ideal generative model would achieve 50% accuracy (although it is possible in theory to generate music which sounds "more human" than human-composed music). We find that LakhNES (Transformer-XL with pre-training) was mistakenly identified as human more often than both our 5-gram model ($p < .0001$) and our LSTM ($p = .07$). It also outperformed the Transformer-XL model without pre-training, but with low confidence ($p = .32$).

Overall, these results suggest that there is still a sizable gap between human-composed and computer-generated chiptunes. We suspect that results would be even less favorable for the generative methods if clips longer than 5 seconds were used. Subjectively speaking, we feel that

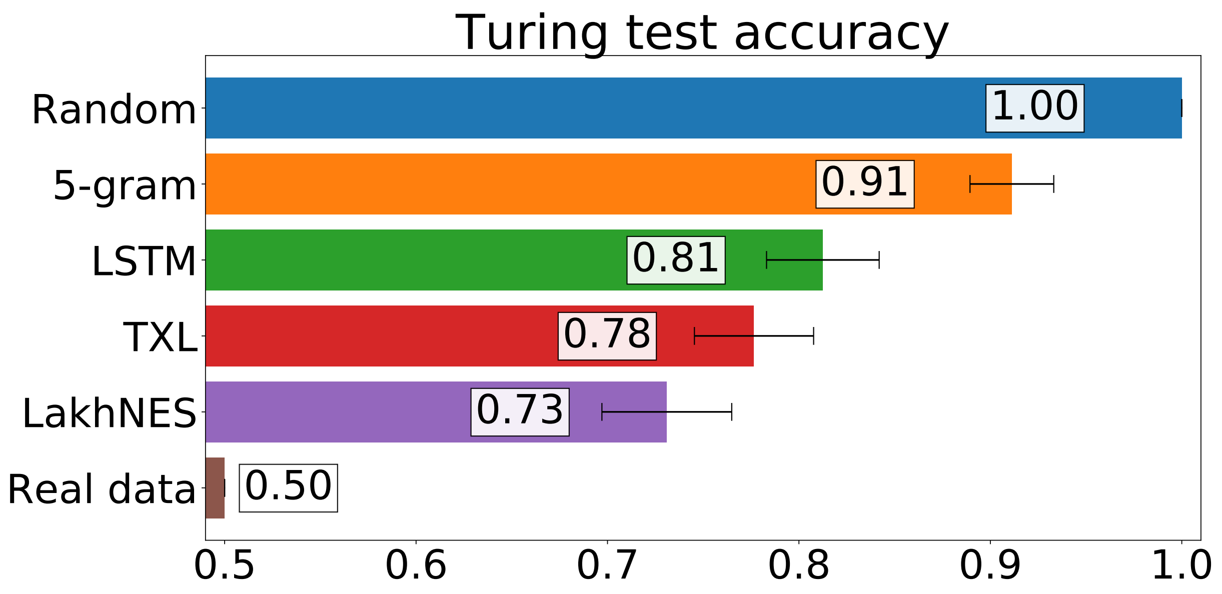


Figure 3.6: Human accuracy at distinguishing computer-generated examples from human-composed ones (error bars are standard error). Users were presented with pairs of clips (one human, one computer) and tasked with identifying which was composed by a human. Random examples are used as a control and we filtered annotators with accuracy less than 1 on those pairs. A *lower* accuracy is better as it indicates that the annotators confused a particular model with the real data more often.

the melodies and harmonies produced by LakhNES are promisingly human, but its inability to maintain rhythmic consistency is often a dead giveaway in a Turing test. We suspect that our model could be improved by using a beat-based event representation, however the current model can be bootstrapped with human-specified rhythmic material to “fix” its rhythmic consistency issues (Section 3.8).

3.7.2 Preference test

In addition to our Turing test, we also conduct a preference-based user study, given that human-ness is not necessarily a predictor of general preference. We present human judges with pairs of examples from two different methods, and ask them which of the two they “prefer”.

Here we construct pairs of 10-second clips from two different (randomly-chosen) methods. As in our Turing test, we construct randomly-ordered batches consisting of 10 pairs. In each

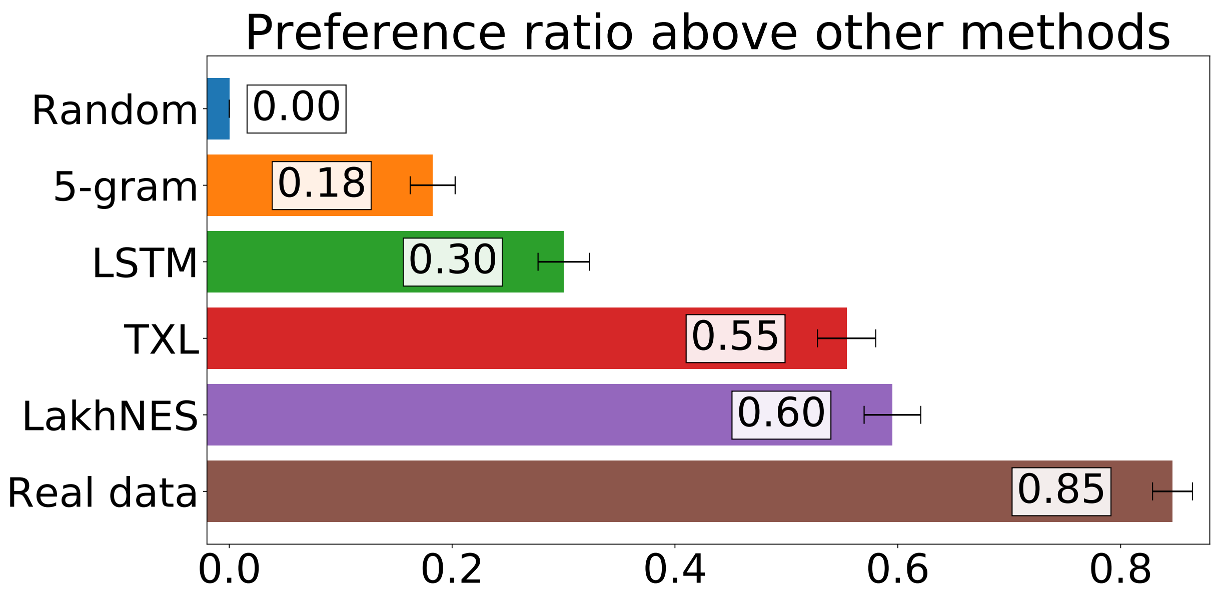


Figure 3.7: Proportion of comparisons where humans preferred an example from each model over an example from another random model (error bars are standard error). Users were presented with pairs of clips from different methods and asked which they preferred. Pairs of random data and human-composed clips are used as a control and we filtered annotators who preferred random. A *higher* ratio is better as it indicates that the annotators preferred results from that method more often than another.

batch, 8 of the pairs are created by sampling two methods without replacement from a set of *five* (four computer-generated and the real data), while 2 pairs always compare randomly-generated clips to real data (control). We ask human judges to assign preference to these batches, filtering out workers who even once indicated that they preferred random examples to real data. After filtering, each of the five methods was involved in around 400 comparisons in total. We report the ratio of "wins" for each method in Figure 3.7, i.e. the proportion of times a method was preferred over any of the other four.

We find that LakhNES outperforms all other generative methods, though is preferred significantly less often than the real data. Human judges preferred chiptunes generated by LakhNES over the real data in 26% of comparisons (vs. only 10% of the time for the LSTM). We find this to be a promising indicator of Transformer's potential on this task.

3.8 Pairing LakhNES with humans

In addition to generating chiptunes from scratch, LakhNES can be used for a number of tasks to assist human composers. For example, LakhNES can be "primed" on human-composed material and then asked to continue the material, providing a method for composers to quickly expand on their ideas. Composers can also provide fixed rhythmic material and use LakhNES to generate the rest of the score. We explore these use cases in our sound examples: https://bit.ly/2PtQTXK . When generating all of our sound examples (besides those in our user study), we found that limiting the entropy of the model by using a sampling temperature of .95 and top-k sampling [Fan et al., 2018] with $k = 32$ improved results qualitatively.

3.9 Conclusion

In this chapter we presented LakhNES, a method for learning to generate multi-instrumental music. We developed an event-based representation suitable for multi-instrumental music. Training powerful language models on this representation results in compelling multi-instrumental music generation. We show that we can further improve results both quantitatively and qualitatively by pre-training on a cross-domain dataset. This cross-domain pre-training procedure could be useful for overcoming data limitations for other musical domains. LakhNES can be used to both generate chiptunes from scratch and collaborate with human composers in a number of different ways.

3.10 Acknowledgements

I would like to thank Louis Pisha for invaluable advice on the technical details of this project. Additionally, I would like to thank Nicolas Boulanger-Lewandowski, Eunjeong Stella Koh, Steven Merity, Miller Puckette, and Cheng-i Wang for helpful conversations throughout this work. This work was supported by UC San Diego’s Chancellors Research Excellence Scholarship program. GPUs used in this research were donated by NVIDIA.

Chapter 3 contains material found in the following two papers. (1) *The NES Music Database: A multi-instrumental dataset with expressive performance attributes*. 2018. Mao, Huanru Henry; McAuley, Julian. International Society for Music Information Retrieval 2018. (2) *LakhNES: Improving multi-instrumental music generation with cross-domain pre-training*. 2019. Mao, Huanru Henry; Li, Yiting Ethan; Cottrell, Garrison W.; McAuley, Julian. Currently under review for publication. The dissertation/thesis author was the primary investigator and author of these papers.

Chapter 4

Music generation in the acoustic domain

Synthesizing audio for specific domains has many practical applications in creative sound design for music and film. Musicians and Foley artists scour large databases of sound effects to find particular audio recordings suitable for specific scenarios. This strategy is painstaking and may result in a negative outcome if the ideal sound effect does not exist in the library. A better approach might allow a sound artist to explore a compact *latent space* of audio, taking broad steps to find the types of sounds they are looking for (e.g. footsteps) and making small adjustments to latent variables to fine-tune (e.g. a large boot lands on a gravel path). However, audio signals have high temporal resolution, and strategies that learn such a representation must perform effectively in high dimensions.

Generative Adversarial Networks (GANs) [Goodfellow et al., 2014] are one such unsupervised strategy for mapping low-dimensional latent vectors to high-dimensional data. The potential advantages of GAN-based approaches to audio synthesis are numerous. Firstly, GANs could be useful for data augmentation [Shrivastava et al., 2017] in data-hungry speech recognition systems. Secondly, GANs could enable rapid and straightforward sampling of large amounts of audio. Furthermore, while the usefulness of generating static images with GANs is arguable, there are many applications (e.g. Foley) for which generating sound effects is immediately useful.

But despite their increasing fidelity at synthesizing images [Radford et al., 2016, Berthelot et al., 2017, Karras et al., 2018], GANs have yet to be demonstrated capable of synthesizing audio in an unsupervised setting.

A naïve solution for applying image-generating GANs to audio would be to operate them on image-like *spectrograms*, i.e., time-frequency representations of audio. This practice of bootstrapping image recognition algorithms for audio tasks is commonplace in the discriminative setting [Hershey et al., 2017]. In the generative setting however, this approach is problematic as the most perceptually-informed spectrograms are non-invertible, and hence cannot be listened to without lossy estimations [Griffin and Lim, 1984] or learned inversion models [Shen et al., 2018].

Recent work [van den Oord et al., 2016, Mehri et al., 2017] has shown that neural networks can be trained with autoregression to operate on *raw audio*. Such approaches are attractive as they dispense with engineered feature representations. However, unlike with GANs, the autoregressive setting results in slow generation as output audio samples must be fed back into the model one at a time.

In this work, we investigate both waveform and spectrogram strategies for generating one-second slices of audio with GANs.[1] For our spectrogram approach (SpecGAN), we first design a spectrogram representation that allows for approximate inversion, and bootstrap the two-dimensional deep convolutional GAN (DCGAN) method [Radford et al., 2016] to operate on these spectrograms. In WaveGAN, our waveform approach, we flatten the DCGAN architecture to operate in one dimension, resulting in a model with the same number of parameters and numerical operations as its two-dimensional analog. With WaveGAN, we provide both a starting point for practical audio synthesis with GANs and a recipe for modifying other image generation methods to operate on waveforms.

We primarily envisage our method being applied to the generation of short sound effects

[1]Sound examples: https://chrisdonahue.com/wavegan_examples
Drum demo: https://chrisdonahue.com/wavegan
Generation with pre-trained models: https://bit.ly/2G8NWpi
Training code: https://github.com/chrisdonahue/wavegan

suitable for use in music and film. For example, we trained a WaveGAN on drums, resulting in a procedural drum machine designed to assist electronic musicians (demo https://chrisdonahue.com/wavegan). However, human evaluation for such domain-specific tasks would require expert listeners. Therefore, we also consider a speech benchmark, facilitating straightforward assessment by human annotators. Specifically, we explore a task where success can easily be judged by any English speaker: generating examples of spoken digits "zero" through "nine".

Though our evaluation focuses on a speech generation task, we note that it is *not* our goal to develop a text-to-speech synthesizer. Instead, our investigation concerns whether unsupervised strategies can learn global structure (e.g. words in speech data) implicit in high-dimensional audio signals without conditioning. Our experiments on speech demonstrate that both WaveGAN and SpecGAN can generate spoken digits that are intelligible to humans. On criteria of sound quality and speaker diversity, human judges indicate a preference for the audio generated by WaveGAN compared to that from SpecGAN.

4.1 GAN Preliminaries

GANs learn mappings from low-dimensional latent vectors $\boldsymbol{z} \in \mathcal{Z}$, i.i.d. samples from known prior P_Z, to points in the space of natural data $\mathcal{X}$. In their original formulation [Goodfellow et al., 2014], a generator $G : \mathcal{Z} \mapsto \mathcal{X}$ is pitted against a discriminator $D : \mathcal{X} \mapsto [0,1]$ in a two-player minimax game. G is trained to minimize the following value function, while D is trained to maximize it:

$$V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim P_X}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim P_Z}[\log(1-D(G(\boldsymbol{z})))]. \tag{4.1}$$

In other words, D is trained to determine if an example is real or fake, and G is trained to fool the discriminator into thinking its output is real. Goodfellow et al. [2014] demonstrate that their proposed training algorithm for Equation 4.1 equates to minimizing the Jensen-Shannon divergence between P_X, the data distribution, and P_G, the implicit distribution of the generator when $\boldsymbol{z} \sim P_Z$. In this original formulation, GANs are notoriously difficult to train, and prone to catastrophic failure cases. Instead of Jensen-Shannon divergence, Arjovsky et al. [2017] suggest minimizing the smoother Wasserstein-1 distance between generated and data distributions

$$W(P_X, P_G) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_X}[f(x)] - \mathbb{E}_{x \sim P_G}[f(x)] \tag{4.2}$$

where $\|f\|_L \leq 1 : \mathcal{X} \mapsto \mathbb{R}$ is the family of functions that are 1-Lipschitz.

To minimize Wasserstein distance, they suggest a GAN training algorithm (WGAN), similar to that of Goodfellow et al. [2014], for the following value function:

$$V_{\text{WGAN}}(D_w, G) = \mathbb{E}_{\boldsymbol{x} \sim P_X}[D_w(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z} \sim P_Z}[D_w(G(\boldsymbol{z}))]. \tag{4.3}$$

With this formulation, $D_w : \mathcal{X} \mapsto \mathbb{R}$ is not trained to identify examples as real or fake, but instead is trained as a function that assists in computing the Wasserstein distance. Arjovsky et al. [2017] suggest weight clipping as a means of enforcing that D_w is 1-Lipschitz. As an alternative strategy, Gulrajani et al. [2017] replace weight clipping with a gradient penalty (WGAN-GP) that also enforces the constraint. They demonstrate that their WGAN-GP strategy can successfully train a variety of model configurations where other GAN losses fail.

4.2 WaveGAN

We motivate our design choices for WaveGAN by first highlighting the different types of structure found in audio versus images.

4.2.1 Intrinsic differences between audio and images

One way to illustrate the differences between audio and images is by examining the axes along which these types of data vary most substantially, i.e. by principal component analysis.

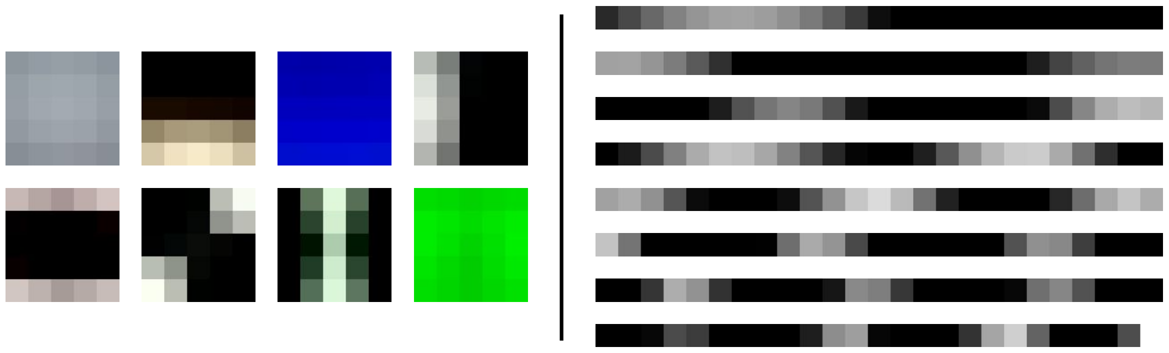


Figure 4.1: First eight principal components for 5x5 patches from natural images (**left**) versus those of length-25 audio slices from speech (**right**). Periodic patterns are unusual in natural images but a fundamental structure in audio.

In Figure 4.1, we show the first eight principal components for patches from natural images and slices from speech. While the principal components of images generally capture intensity, gradient, and edge characteristics, those from audio form a periodic basis that decompose the audio into constituent frequency bands. In general, natural audio signals are more likely to exhibit periodicity than natural images.

As a consequence, correlations across large windows are commonplace in audio. For example, in a waveform sampled at 16kHz, a 440Hz sinusoid (the musical note A4) takes over 36 samples to complete a single cycle. This suggests that filters with larger receptive fields are needed to process raw audio. This same intuition motivated van den Oord et al. [2016] in their design of WaveNet, which uses dilated convolutions to exponentially increase the model's effective receptive field with linear increase in layer depth.

4.2.2 WaveGAN architecture

We base our WaveGAN architecture off of DCGAN [Radford et al., 2016] which popularized usage of GANs for image synthesis. The DCGAN generator uses the *transposed convolution* operation (Figure 4.2) to iteratively upsample low-resolution feature maps into a high-resolution image. Motivated by our above discussion, we modify this transposed convolution operation to

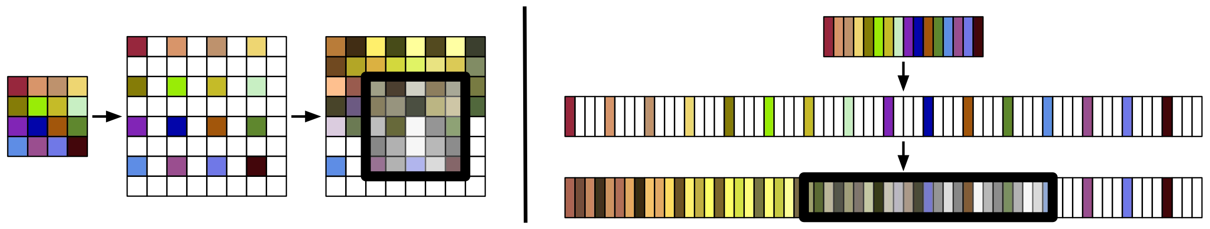


Figure 4.2: Depiction of the transposed convolution operation for the first layers of the DCGAN [Radford et al., 2016] (**left**) and WaveGAN (**right**) generators. DCGAN uses small (5x5), two-dimensional filters while WaveGAN uses longer (length-25), one-dimensional filters and a larger upsampling factor. Both strategies have the same number of parameters and numerical operations.

widen its receptive field. Specifically, we use longer one-dimensional filters of length 25 instead of two-dimensional filters of size 5x5, and we upsample by a factor of 4 instead of 2 at each layer (Figure 4.2). We modify the discriminator in a similar way, using length-25 filters in one dimension and increasing stride from 2 to 4. These changes result in WaveGAN having the same number of parameters, numerical operations, and output dimensionality as DCGAN.

Because DCGAN outputs 64x64 pixel images — equivalent to just 4096 audio samples — we add one additional layer to the model resulting in 16384 samples, slightly more than one second of audio at 16 kHz. This length is already sufficient for certain sound domains (e.g. sound effects, voice commands), and future work adapting megapixel image generation techniques [Karras et al., 2018] could expand the output length to more than a minute. We requantize the real data from its 16-bit integer representation (linear pulse code modulation) to 32-bit floating point, and our generator similarly outputs floating point waveforms. A complete description of our model is in [Donahue et al., 2019a].

In summary, we outline our modifications to the DCGAN [Radford et al., 2016] method which result in WaveGAN. This straightforward recipe already produces reasonable audio, and further contributions outlined below and in Appendix 4.3 serve to refine results.

1. Flatten 2D convolutions into 1D (e.g. 5x5 2D convolution becomes length-25 1D).

2. Increase the stride factor for all convolutions (e.g. stride 2x2 becomes stride 4).
3. Remove batch normalization from the generator and discriminator.
4. Train using the WGAN-GP [Gulrajani et al., 2017] strategy.

4.3 Understanding and mitigating artifacts in generated audio

Generative models that upsample by transposed convolution are known to produce characteristic "checkerboard" artifacts in images [Odena et al., 2016], artifacts with particular spatial periodicities. The discriminator of image-generating GANs can learn to reject images with these artifacts because they are uncommon in real data (as discussed in Section 4.2.1). However, in the audio domain, the discriminator might not have such luxury as these artifacts correspond to frequencies which might rightfully appear in the real data.

While checkerboard artifacts are an annoyance in image generation, they can be devastating to audio generation results. While our eye may perceive these types of periodic distortions as an intrusive texture, our ear perceives them as an abrasive tone. To characterize these artifacts in WaveGAN, we measure its impulse response by randomly initializing it 1000 times and passing unit impulses to its first convolutional layer. In Figure 4.3, we plot the average of these responses in the frequency domain. The response has sharp peaks at linear multiples of the sample rates of each convolutional layer (250Hz, 1kHz, 4kHz, etc.). This is in agreement with our informal observation of results from WaveGAN, which often have a pitched noise close to the musical note B (247×2^nHz).

Below, we will discuss strategies we designed to mitigate these artifacts in WaveGAN.

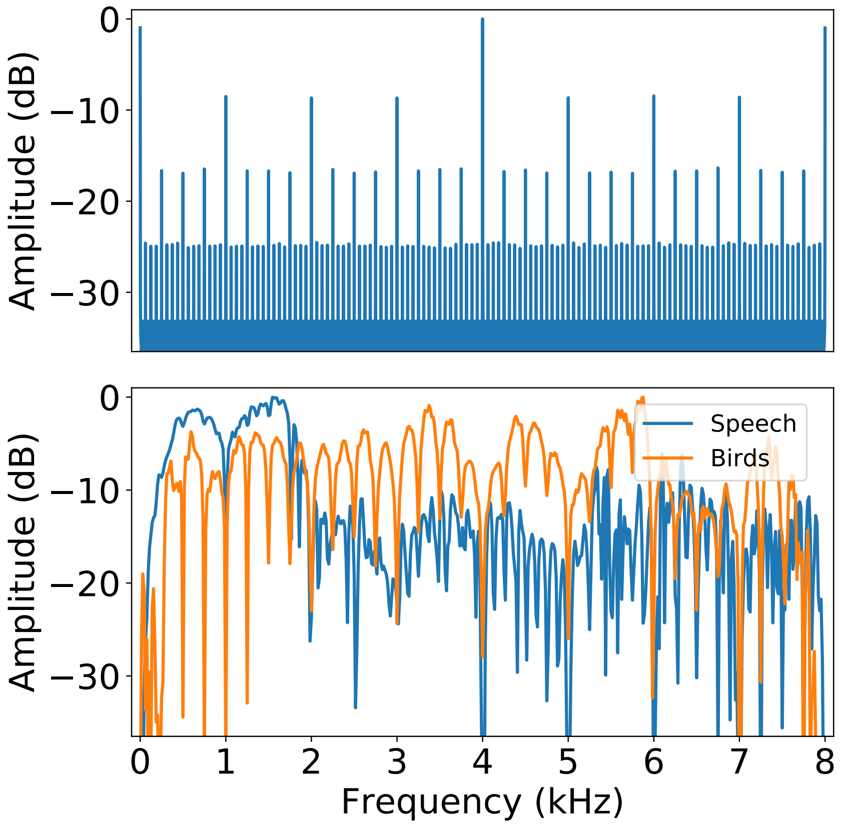


Figure 4.3: (**Top**): Average impulse response for 1000 random initializations of the WaveGAN generator. (**Bottom**): Response of learned post-processing filters for speech and bird vocalizations. Post-processing filters reject frequencies corresponding to noise byproducts created by the generative procedure (top). The filter for speech boosts signal in prominent speech bands, while the filter for bird vocalizations (which are more uniformly-distributed in frequency) simply reduces noise presence.

4.3.1 Learned post-processing filters

We experiment with adding a post-processing filter to the generator, giving WaveGAN a simple mechanism to filter out undesirable frequencies created by the generative process. This filter has a long window (512 samples) allowing it to represent intricate transfer functions, and the weights of the filter are learned as part of the generator's parameters. In Figure 4.3, we compare the post-processing filters that WaveGAN learns for human speech and bird vocalizations. The filters boost signal in regions of the frequency spectrum that are most prominent in the real data

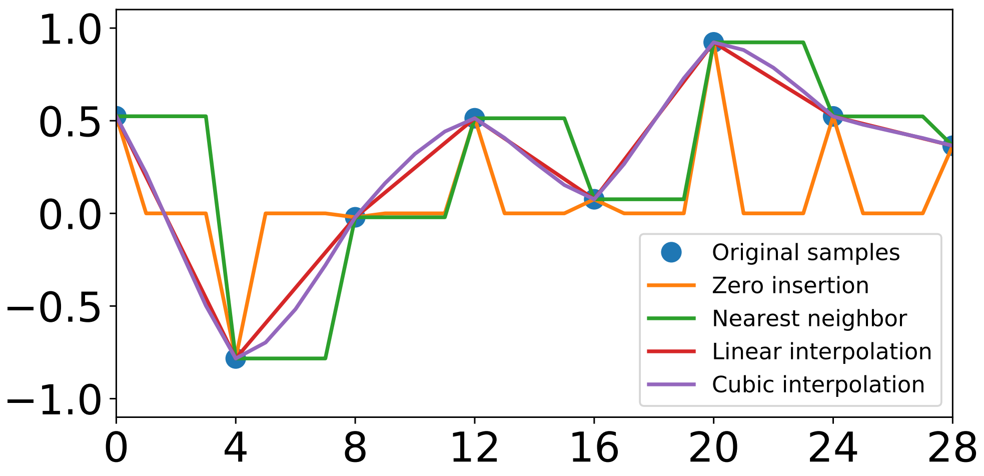


Figure 4.4: Depiction of the upsampling strategy used by transposed convolution (zero insertion) and other strategies which mitigate aliasing: nearest neighbor, linear and cubic interpolation.

domain, and introduce notches at bands that are artifacts of the generative procedure as discussed in the previous section.

4.3.2 Upsampling procedure

Transposed convolution upsamples signals by inserting zeros in between samples and applying a learned filterbank. This operation introduces aliased frequencies, copies of pre-existing frequencies shifted by multiples of the new Nyquist rate, into the upsampled signal. While aliased frequencies are usually seen as undesirable artifacts of a bad upsampling procedure, in the generative setting their existence may be crucial for producing fine-grained details in the output.

We experiment with three other upsampling strategies in WaveGAN: nearest-neighbor, linear and cubic interpolation, all of which attenuate aliased frequencies. In Figure 4.4, we compare these strategies visually. While nearest neighbor upsampling resulted in similar audio output to transposed convolution, linear and cubic interpolation strategies resulted in qualitatively poor audio output (sound examples: https://chrisdonahue.com/wavegan_examples). We hypothesize that the aliased frequencies produced by upsampling convolutions may be more critical to audio generation than image generation.

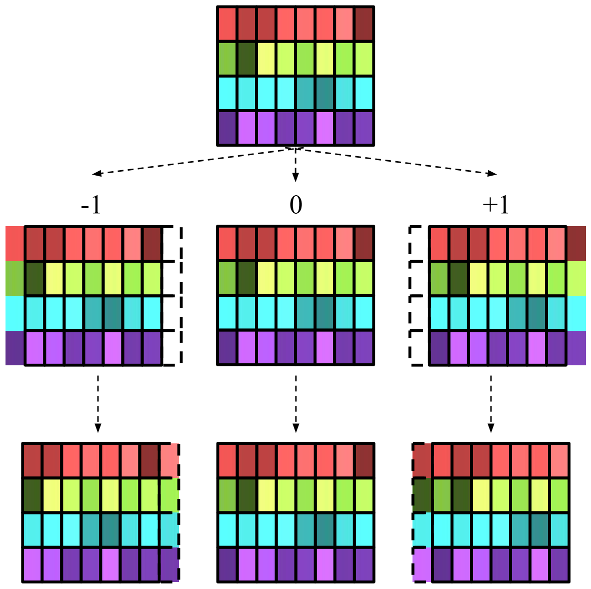


Figure 4.5: At each layer of the WaveGAN discriminator, the phase shuffle operation perturbs the phase of each feature map by Uniform $\sim [-n, n]$ samples, filling in the missing samples (dashed outlines) by reflection. Here we depict all possible outcomes for a layer with four feature maps ($n = 1$).

4.3.3 Phase shuffle

Generative image models that upsample by transposed convolution (such as DCGAN) are known to produce characteristic "checkerboard" artifacts in images [Odena et al., 2016]. Periodic patterns are less common in images (Section 4.2.1), and thus the discriminator can learn to reject images that contain them. For audio, analogous artifacts are perceived as pitched noise which may overlap with frequencies commonplace in the real data, making the discriminator's objective more challenging. However, the artifact frequencies will always occur at a particular phase, allowing the discriminator to learn a trivial policy to reject generated examples. This may inhibit the overall optimization problem.

To prevent the discriminator from learning such a solution, we propose the *phase shuffle* operation with hyperparameter n. Phase shuffle randomly perturbs the phase of each layer's activations by $-n$ to n samples before input to the next layer (Figure 4.5). We apply phase shuffle only to the discriminator, as the latent vector already provides the generator a mechanism to manipulate the phase of a resultant waveform. Intuitively speaking, phase shuffle makes the discriminator's job more challenging by requiring invariance to the phase of the input waveform.

4.4 SpecGAN: Generating semi-invertible spectrograms

While a minority of recent research in discriminative audio classification tasks has used raw audio input [Sainath et al., 2015, Lee et al., 2017], most of these approaches operate on spectrogram representations of audio. A generative model may also benefit from operating in such a time-frequency space. However, commonly-used representations in the discriminative setting are uninvertible.

With SpecGAN, our frequency-domain audio generation model, we design a spectrogram representation that is both well-suited to GANs designed for image generation and *can* be approximately inverted. Additionally, to facilitate direct comparison, our representation is designed to use the same dimensionality per unit of time as WaveGAN (16384 samples yield a 128x128 spectrogram).

To process audio into suitable spectrograms, we first perform the short-time Fourier transform with 16ms windows and 8ms stride, resulting in 128 frequency bins[2] linearly spaced from 0 to 8kHz. We take the magnitude of the resultant spectra and scale amplitude values logarithmically to better-align with human perception. We then normalize each frequency bin to have zero mean and unit variance. This type of preprocessing is commonplace in audio classification, but produce spectrograms with unbounded values—a departure from image representations. We therefore clip the spectra to 3 standard deviations and rescale to $[-1, 1]$. Through an informal listening test, we determined that this clipping strategy did not produce an audible difference during inversion.

Once our dataset has been processed into this format, we operate the DCGAN [Radford

[2]The FFT for this window size actually produces 129 frequency bins. We discard the top (Nyquist) bin from each example for training. During resynthesis, we replace it with the dataset's mean for that bin.

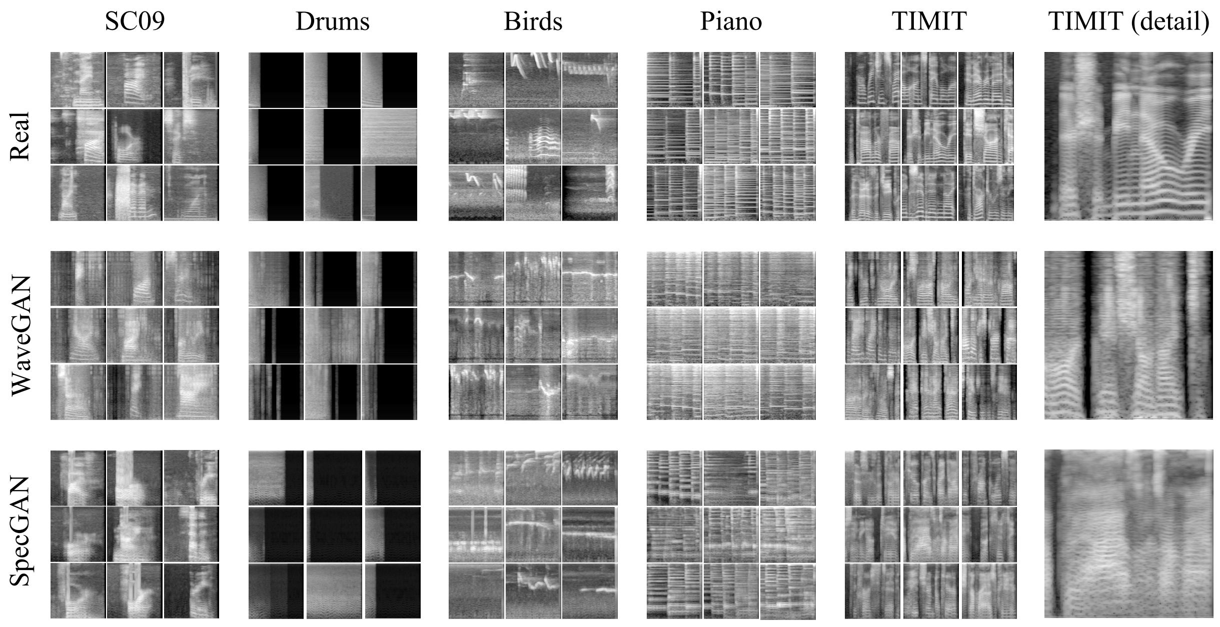


Figure 4.6: **Top**: Random samples from each of the five datasets used in this study, illustrating the wide variety of spectral characteristics. **Middle**: Random samples generated by WaveGAN for each domain. WaveGAN operates in the time domain but results are displayed here in the frequency domain for visual comparison. **Bottom**: Random samples generated by SpecGAN for each domain.

et al., 2016] algorithm on the resultant spectra. To render the resultant generated spectrograms as waveforms, we first invert the steps of spectrogram preprocessing described above, resulting in linear-amplitude magnitude spectra. We then employ the iterative Griffin-Lim algorithm [Griffin and Lim, 1984] with 16 iterations to estimate phase and produce 16384 audio samples.

4.5 Experimental protocol

To facilitate human evaluation, our experimentation focuses on the *Speech Commands Dataset* [Warden, 2018]. This dataset consists of many speakers recording individual words in uncontrolled recording conditions. We explore a subset consisting of the spoken digits "zero" through "nine" and refer to this subset as the *Speech Commands Zero Through Nine* (SC09) dataset. While this dataset is intentionally reminiscent of the popular MNIST dataset of written digits, we note that examples from SC09 are much higher dimensional ($\mathbb{R}^{16000}$) than examples from MNIST ($\mathbb{R}^{28\times28=784}$).

These ten words encompass many phonemes and two consist of multiple syllables. Each recording is one second in length, and we do not attempt to align the words in time. There are 1850 utterances of each word in the training set, resulting in 5.3 hours of speech. The heterogeneity of alignments, speakers, and recording conditions make this a challenging dataset for generative modeling.

Our baseline configuration for WaveGAN excludes phase shuffle. We compare this to the performance of WaveGAN with phase shuffle ($n \in \{2,4\}$) and a variant of WaveGAN which uses nearest-neighbor upsampling rather than transposed convolution [Odena et al., 2016]. Hoping to reduce noisy artifacts, we also experiment with adding a wide (length-512) post-processing filter to the output of the generator and learning its parameters with the rest of the generator variables (details in Appendix 4.3.1). We use the WGAN-GP [Gulrajani et al., 2017] algorithm for all experiments, finding it to produce reasonable results where others [Radford et al., 2016, Mao et al., 2017, Arjovsky et al., 2017] failed. We compare the performance of these configurations to that of SpecGAN.

We also perform experiments on four other datasets with different characteristics (Figure 4.6):

1. *Drum sound effects* (0.7 hours): Drum samples for kicks, snares, toms, and cymbals
2. *Bird vocalizations* (12.2 hours): In-the-wild recordings of many species [Boesman, 2018]
3. *Piano* (0.3 hours): Professional performer playing a variety of Bach compositions
4. *Large vocab speech (TIMIT)* (2.4 hours): Multiple speakers, clean [Garofolo et al., 1993]

We train our networks using batches of size 64 on a single NVIDIA P100 GPU. During our quantitative evaluation of SC09 (discussed below), our WaveGAN networks converge by their early stopping criteria (inception score) within four days (200k iterations, around 3500 epochs), and produce speech-like audio within the first hour of training. Our SpecGAN networks converge more quickly, within two days (around 1750 epochs). On the other four datasets, we train WaveGAN for 200k iterations representing nearly 1500 epochs for the largest dataset. Unlike with autoregressive methods [van den Oord et al., 2016, Mehri et al., 2017], generation with WaveGAN is fully parallel and can produce an hour of audio in less than two seconds. We list all hyperparameters in [Donahue et al., 2019a].

4.6 Evaluation methodology

Evaluation of generative models is a fraught topic. Theis et al. [2016] demonstrate that quantitative measures of sample quality are poorly correlated with each other and human judgement. Accordingly, we use several quantitative evaluation metrics for hyperparameter validation and discussion, and also evaluate our most promising models with human judges.

4.6.1 Inception score

Salimans et al. [2016] propose the *inception score*, which uses a pre-trained Inception classifier [Szegedy et al., 2016] to measure both the diversity and semantic discriminability of generated images, finding that the measure correlates well with human judgement.

Given model scores $P(\boldsymbol{y} \mid \boldsymbol{x})$ with marginal $P(\boldsymbol{y})$, the inception score is defined as $\exp(\mathbb{E}_{\boldsymbol{x}} D_{\mathrm{KL}}(P(\boldsymbol{y} \mid \boldsymbol{x}) || P(\boldsymbol{y})))$, and is estimated over a large number of samples (e.g. 50k). For n classes, this measure ranges from 1 to n, and is maximized when the model is completely confident about each prediction *and* predicts each label equally often. We will use this measure as our primary quantitative evaluation method and early stopping criteria.

To measure inception score, we train an audio classifier on SC09. Our classifier first computes a short-time Fourier transform of the input audio with 64ms windows and 8ms stride.

This representation is projected to 128 frequency bins equally spaced on the Mel scale [Stevens et al., 1937] from 40Hz to 7800Hz. Amplitudes are scaled logarithmically and normalized so that each bin has zero mean and unit variance. We process this perceptually-informed representation with four layers of convolution and pooling, projecting the result to a *softmax* layer with 10 classes. We perform early stopping on the minimum negative log-likelihood of the validation set; the resultant model achieves 93% accuracy on the test set. Because this classifier observes spectrograms, our spectrogram-generating models may have a representational advantage over our waveform-generating models.

4.6.2 Nearest neighbor comparisons

Inception score has two trivial failure cases in which a poor generative model can achieve a high score. Firstly, a generative model that outputs a single example of each class with uniform probability will be assigned a high score. Secondly, a generative model that overfits the training data will achieve a high score simply by outputting examples on which the classifier was trained.

We use two indicators metrics to determine if a high inception score has been caused by either of these two undesirable cases. Our first indicator, $|D|_{\text{self}}$, measures the average Euclidean distance of a set of 1k examples to their nearest neighbor within the set (other than itself). A higher $|D|_{\text{self}}$ indicates higher diversity amongst samples. Because measuring Euclidean distance in time-domain audio poorly represents human perception, we evaluate distances in the same frequency-domain representation as our classifier from Section 4.6.1.

Our second indicator, $|D|_{\text{train}}$, measures the average Euclidean distance of 1k examples to their nearest neighbor in the real training data. If the generative model simply produces examples from the training set, this measure will be 0. We report $|D|_{\text{train}}$ and $|D|_{\text{self}}$ relative to those of the test set.

4.6.3 Qualitative human judgements

While inception score is a useful metric for hyperparameter validation, our ultimate goal is to produce examples that are intelligible to humans. To this end, we measure the ability of human annotators on *Amazon Mechanical Turk* to label the generated audio. Using our best WaveGAN and SpecGAN models as measured by inception score, we generate random examples until we have 300 for each digit (as labeled by our classifier from Section 4.6.1)—3000 total. In batches of ten random examples, we ask annotators to label which digit they perceive in each example, and compute their accuracy with respect to the classifier's labels (random accuracy would be 10%). After each batch, annotators assign subjective values of 1–5 for criteria of sound quality, ease of intelligibility, and speaker diversity. We report accuracy ($n = 3000$) and mean opinion scores ($n = 300$) in Table 4.1.

4.7 Results and discussion

Results for our evaluation appear in Table 4.1. We also evaluate our metrics on the real training data, the real test data, and a version of SC09 generated by a parametric speech synthesizer [Buchner, 2017]. We also compare to SampleRNN [Mehri et al., 2017] and two public implementations of WaveNet [van den Oord et al., 2016], but neither method produced competitive results (details in [Donahue et al., 2019a]), and we excluded them from further evaluation. These autoregressive models have not previously been examined on small-vocabulary speech data, and their success at generating full words has only been demonstrated when conditioning on rich linguistic features. Sound examples for all experiments can be found at https://chrisdonahue.com/wavegan_examples.

While the maximum inception score for SC09 is 10, any score higher than the test set score of 8 should be seen as evidence that a generative model has overfit. Our best WaveGAN model uses phase shuffle with $n = 2$ and achieves an inception score of 4.7. To compare the

Table 4.1: Quantitative and qualitative (human study) results for SC09 experiments comparing real and generated data. A higher inception score suggests that semantic modes of the real data distribution have been captured. $|D|_{\text{self}}$ indicates the intra-dataset diversity relative to that of the real test data. $|D|_{\text{train}}$ indicates the distance between the dataset and the training set relative to that of the test data; a low value indicates a generative model that is overfit to the training data. Acc. is the overall accuracy of humans on the task of labeling class-balanced digits (random chance is 0.1). Sound *quality*, *ease* of intelligibility and speaker *diversity* are mean opinion scores (1-5); higher is better.

	Quantitative			*Qualitative (human judges)*			
Experiment	Inception score	$\|D\|_{\text{self}}$	$\|D\|_{\text{train}}$	Acc.	Quality	Ease	Diversity
Real (train)	9.18 ± 0.04	1.1	0.0				
Real (test)	8.01 ± 0.24	1.0	1.0	0.95	3.9 ± 0.8	3.9 ± 1.1	3.5 ± 1.0
Parametric	5.02 ± 0.06	0.7	1.1				
WaveGAN	4.12 ± 0.03	1.4	2.0				
+ Phase shuffle $n = 2$	4.67 ± 0.01	0.8	2.3	0.58	2.3 ± 0.9	2.8 ± 0.9	3.2 ± 0.9
+ Phase shuffle $n = 4$	4.54 ± 0.03	1.0	2.3				
+ Nearest neighbor	3.77 ± 0.02	1.8	2.6				
+ Post-processing	3.92 ± 0.03	1.4	2.9				
+ Dropout	3.93 ± 0.03	1.0	2.6				
SpecGAN	6.03 ± 0.04	1.1	1.4	0.66	1.9 ± 0.8	2.8 ± 0.9	2.6 ± 1.0
+ Phase shuffle $n = 1$	3.71 ± 0.03	0.8	1.6				

effect of phase shuffle to other common regularizers, we also tried using 50% dropout in the discriminator's activations, which resulted in a lower score. Phase shuffle decreased the inception score of SpecGAN, possibly because the operation has an exaggerated effect when applied to the compact temporal axis of spectrograms.

Most experiments produced $|D|_{\text{self}}$ (diversity) values higher than that of the test data, and all experiments produced $|D|_{\text{train}}$ (distance from training data) values higher than that of the test data. While these measures indicate that our generative models produce examples with statistics that deviate from those of the real data, neither metric indicates that the models achieve high inception scores by the trivial solutions outlined in Section 4.6.2.

Compared to examples from WaveGAN, examples from SpecGAN achieve higher inception score (6.0 vs. 4.7) and are labeled more accurately by humans (66% vs. 58%). However, on subjective criteria of sound quality and speaker diversity, humans indicate a preference for examples from WaveGAN. It appears that SpecGAN might better capture the variance in the underlying data compared to WaveGAN, but its success is compromised by sound quality issues when its spectrograms are inverted to audio. It is possible that the poor qualitative ratings for examples from SpecGAN are primarily caused by the lossy Griffin-Lim inversion [Griffin and Lim, 1984] and not the generative procedure itself. We see promise in both waveform and spectrogram audio generation with GANs; our study does not suggest a decisive winner. For a more thorough investigation of spectrogram generation methods, we point to follow-up work [Engel et al., 2019].

Finally, we train WaveGAN and SpecGAN models on the four other domains listed in Section 4.5. Somewhat surprisingly, we find that the frequency-domain spectra produced by WaveGAN (a time-domain method) are visually more consistent with the training data (e.g. in terms of sharpness) than those produced by SpecGAN (Figure 4.6). For drum sound effects, WaveGAN captures semantic modes such as kick and snare drums. On bird vocalizations, WaveGAN generates a variety of distinct bird sounds. On piano, WaveGAN produces musically-consonant motifs that, as with the training data, represent a variety of key signatures and rhythmic patterns. For TIMIT, a large-vocabulary speech dataset with many speakers, WaveGAN produces speech-like babbling similar to results from unconditional autoregressive models [van den Oord et al., 2016].

4.8 Related work

Much of the work within generative modeling of audio is within the context of text-to-speech. Text-to-speech systems are primarily either *concatenative* or *parametric*. In concatenative systems, audio is generated by sequencing small, prerecorded portions of speech from a phonetically-indexed dictionary [Moulines and Charpentier, 1990, Hunt and Black, 1996]. Parametric systems map text to salient parameters of speech, which are then synthesized by a vocoder [Dudley, 1939]; see [Zen et al., 2009] for a comprehensive review. Some of these systems use learning-based approaches such as a hidden Markov models [Yoshimura, 2002, Tokuda et al., 2013], and separately-trained neural networks pipelines [Ling et al., 2015] to estimate speech parameters.

Recently, several researchers have investigated parametric speech synthesis with end-to-end neural network approaches that learn to produce vocoder features directly from text or phonetic embeddings [Gibiansky et al., 2017, Ping et al., 2018, Sotelo et al., 2017, Wang et al., 2017, Shen et al., 2018]. These vocoder features are synthesized to raw audio using off-the-shelf methods such as WORLD [Morise et al., 2016] and Griffin-Lim [Griffin and Lim, 1984], or trained neural vocoders [Sotelo et al., 2017, Shen et al., 2018, Ping et al., 2018]. All of these methods are supervised: they are trained to map linguistic features to audio outputs.

Several approaches have explored unsupervised generation of raw audio. van den Oord et al. [2016] propose WaveNet, a convolutional model which learns to predict raw audio samples by autoregressive modeling. WaveNets conditioned on rich linguistic features have widely been deployed in text-to-speech systems, though they have not been demonstrated capable of generating cohesive words in the unconditional setting. Engel et al. [2017] pose WaveNet as an autoencoder to generate musical instrument sounds. Chung et al. [2014], Mehri et al. [2017] both train recurrent autoregressive models which learn to predict raw audio samples. While autoregressive methods generally produce higher audio fidelity than WaveGAN, synthesis with WaveGAN is orders of magnitude faster.

The application of GANs [Goodfellow et al., 2014] to audio has so far been limited to supervised learning problems in combination with traditional loss functions. Pascual et al. [2017] apply GANs to raw audio speech enhancement. Their encoder-decoder approach combines the GAN objective with an L_2 loss. Fan et al. [2017], Michelsanti and Tan [2017], Donahue et al. [2018a] all use GANs in combination with unstructured losses to map spectrograms in one domain to spectrograms in another. Chen et al. [2017] use GANs to map musical performance images into spectrograms.

4.9 Improving SpecGAN with adversarial vocoding

Generating natural-sounding speech from text is a well-studied problem with numerous potential applications. While past approaches were built on extensive engineering knowledge in the areas of linguistics and speech processing (see [Zen et al., 2009] for a review), recent approaches adopt neural network strategies which learn from data to map linguistic representations into audio waveforms [Arik et al., 2017, Gibiansky et al., 2017, Ping et al., 2018, Wang et al., 2017, Shen et al., 2018]. Of these recent systems, the best performing [Ping et al., 2018, Shen et al., 2018] are both comprised of two functional mechanisms which (1) map language into *perceptually-informed spectrogram* representations (i.e., time-frequency decompositions of audio with logarithmic scaling of both frequency and amplitude), and (2) *vocode* the resultant spectrograms into listenable waveforms. In such two-step TTS systems, using perceptually-informed spectrograms as intermediaries is observed to have empirical benefits over using representations which are simpler to convert to audio [Ping et al., 2018]. Hence, vocoding is central to the success of state-of-the-art TTS systems, and is the focus of this work.

The need for vocoding arises from the non-invertibility of perceptually-informed spectrograms. These compact representations exclude much of the information in an audio waveform, and thus require a predictive model to fill in the missing information needed to synthesize natural-sounding audio. Notably, standard spectrogram representations discard phase information resulting from the short-time Fourier transform (STFT), and additionally compress the linearly-scaled frequency axis of the STFT magnitude spectrogram into a logarithmically-scaled one. This gives rise to two corresponding vocoding subproblems: the well-known problem of *phase estimation*, and the less-investigated problem of *magnitude estimation*.

Vocoding methodology in state-of-the-art TTS systems [Ping et al., 2018, Shen et al., 2018] endeavors to address the joint of these two subproblems, i.e., to transform perceptually-informed spectrograms directly into waveforms. Specifically, both systems use WaveNet [van den Oord et al., 2016] conditioned on spectrograms. This approach is problematic as it necessitates running WaveNet once per individual audio sample (e.g. 22050 times per second), bottlenecking the overall TTS system as the language-to-spectrogram mechanisms are comparatively fast.[3] Given that joint solutions currently necessitate such computational overhead, it may be methodologically advantageous to combine solutions to the individual subproblems.

Before endeavoring to develop individual solutions to magnitude and phase estimation, we first wished to discover which (if any) of the two represented a greater obstacle to vocoding. To answer this, we conducted a user study examining the effect that common heuristics for each subproblem have on the perceived naturalness of vocoded speech (Table 4.2).[4] Our study demonstrated that **combining an ideal solution to *either* magnitude or phase estimation with a heuristic for the other results in high-quality speech**. Hence, we can focus our research efforts on *either* subproblem, in the hopes of developing methods which are more computationally efficient than existing end-to-end strategies.

In this paper, we seek to address the magnitude estimation subproblem, which is well-suited for modern deep learning methodology. We propose a learning-based method which uses Generative Adversarial Networks [Goodfellow et al., 2014] to learn a stochastic mapping from perceptually-informed spectrograms into simple magnitude spectrograms. We combine this magnitude estimation method with a modern phase estimation heuristic, referring to this method as *adversarial vocoding*. We show that adversarial vocoding can be used to expedite TTS synthesis and additionally improves upon the state-of-the-art in unsupervised generation of individual words of speech.

[3]In our empirical experimentation with open-source codebases, the autoregressive vocoding phase was over 1500 times slower on average than the language to spectrogram phase.

[4]Sound examples: https://chrisdonahue.com/advoc_examples

4.9.1 Summary of contributions

- We measure the perceived effect of inverting the primary sources of compression in audio features. We observe that coupling solutions to either compression source with a heuristic for the other result in high-quality speech.
- For both real spectrograms and synthetic ones from TTS systems, we demonstrate that our proposed vocoding method yields significantly higher mean opinion scores than a heuristic baseline and faster speeds than state-of-the-art vocoding methods.
- We show that our method can effectively vocode highly-compressed (13:1) audio feature representations.
- We show that our method improves the state of the art in unsupervised synthesis of individual words of speech.

4.10 Audio feature preliminaries

The typical process of transforming waveforms into perceptually-informed spectrograms involves several cascading stages. Here, we describe spectrogram methodology common to two state-of-the-art TTS systems [Ping et al., 2018, Shen et al., 2018]. A visual representation is shown in Figure 4.7.

Extraction The initial stage consists of decomposing waveforms into time and frequency using the STFT. Then, the phase information is discarded from the complex STFT coefficients leaving only the linear-amplitude magnitude spectrogram. The linearly-spaced frequency bins of the resultant spectrogram are then compressed to fewer bins which are equally-spaced on a logarithmic scale (usually the mel scale [Stevens et al., 1937]). Finally, amplitudes of the resultant spectrogram are made logarithmic to conform to human loudness perception, then optionally clipped and normalized.

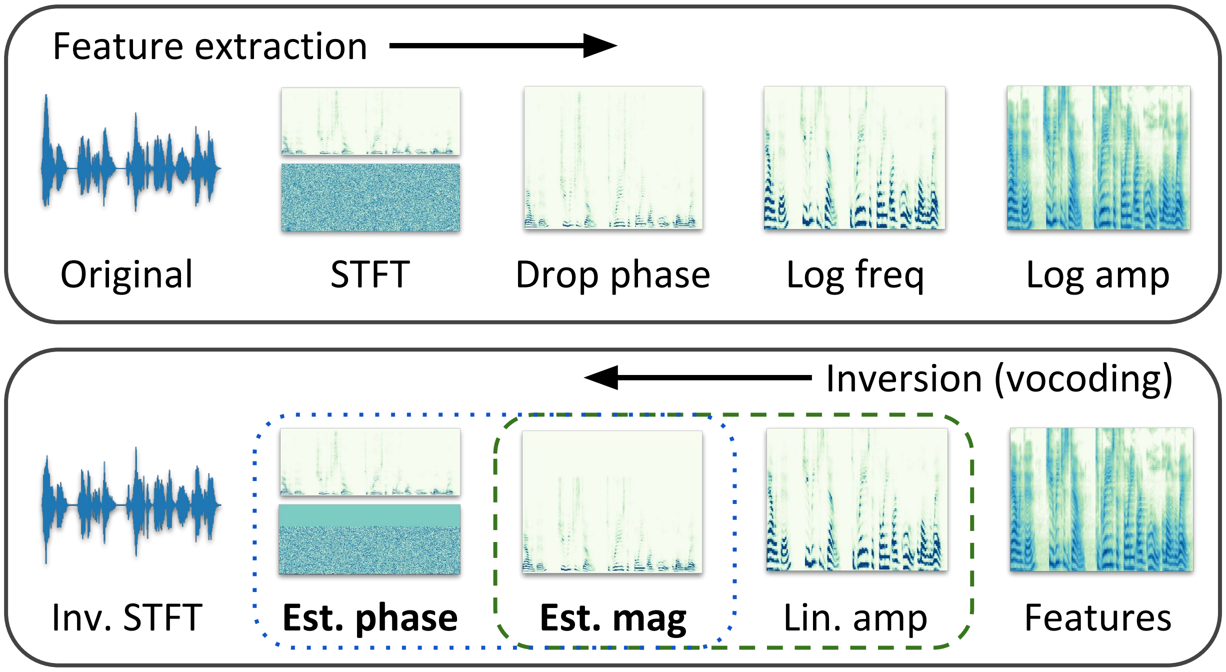


Figure 4.7: Depiction of stages in common audio feature extraction pipelines and corresponding inversion. The two obstacles to vocoding are (1) estimating linear-frequency magnitude spectra from log-frequency mel spectra (outlined in green dashed line), and (2) estimating phase information from magnitude spectra (outlined in blue dotted line). We focus on magnitude estimation in this paper, observing that coupling an ideal solution to this subproblem with a phase estimation heuristic can produce high-quality speech (Table 4.2).

Inversion To heuristically invert this procedure (vocode), the inverse of each cascading step is applied in reverse. First, logarithmic amplitudes are converted to linear ones. Then, an appropriate magnitude spectrogram is estimated from the mel spectrogram. Finally, appropriate phase information is estimated from the magnitude spectrogram, and the inverse STFT is used to render audio.

Unless otherwise specified, throughout this paper we operate on waveforms sampled at 22050Hz using an STFT with a window size of 1024 and a hop size of 256. We compress magnitude spectrograms to 80 bins ($melBins = 80$) equally spaced along the mel scale from 125Hz to 7600Hz. We apply log amplitude scaling and normalize resultant mel spectrograms to have 120dB dynamic range. Precisely recreating this representation [McFee et al., 2019] is simple in our codebase.[5]

4.11 Measuring the effect of magnitude and phase estimation on speech naturalness

The audio feature extraction pipelines outlined in Section 4.10 have two sources of compression: the discarding of phase information and compression of magnitude information. Conventional wisdom suggests that the primary obstacle to inverting such features is phase estimation. However, to the best of our knowledge, a systematic evaluation of the individual contributions of magnitude and phase estimation on perceived naturalness of vocoded speech has never been conducted.

To perform such an evaluation, we mix and match methods for estimating both STFT magnitudes and phases from log-amplitude mel spectrograms. A common heuristic for magnitude estimation is to project the mel-scale spectrogram onto the pseudoinverse of the mel basis which was originally used to generate it. As a phase estimation baseline, state-of-the-art TTS research [Ping et al., 2018, Shen et al., 2018] compares to the iterative Griffin-Lim [Griffin and Lim, 1984] strategy with 60 iterations. We additionally consider the more-recent Local Weighted Sums (LWS) [Le Roux et al., 2010] strategy which, on our CPU, is about six times faster than 60 iterations of Griffin-Lim. As a proxy for an ideal solution to either subproblem, we also use magnitude and phase information extracted from real data.

We show human judges the same waveform vocoded by six different magnitude and phase estimation combinations (inducing a comparison) and ask them to rate the naturalness of each on a subjective 1 to 5 scale (full user study methodology outlined in Section 4.13.1). Mean opinion scores are shown in Table 4.2, and we encourage readers to listen to our sound examples linked from the footnote on the first page to help contextualize.

[5]Code: https://github.com/paarthneekhara/advoc

Table 4.2: Ablating the effect of heuristics for magnitude and phase estimation on mean opinion score (MOS) of speech naturalness with 95% confidence intervals. **Bolded** entries show that coupling an ideal solution to either subproblem (real data used as a proxy) with a good heuristic for the other yields speech with only 2–9% lower MOS than real speech ($p < 0.05$).

Magnitude est. method	Phase est. method	MOS
Ideal (real magnitudes)	*Ideal* (real phases)	4.30 ± 0.06
Ideal (real magnitudes)	Griffin-Lim w/ 60 iters	3.70 ± 0.07
Ideal (real magnitudes)	Local Weighted Sums	$\mathbf{4.09 \pm 0.06}$
Mel pseudoinverse	*Ideal* (real phases)	$\mathbf{4.04 \pm 0.06}$
Mel pseudoinverse	Griffin-Lim w/ 60 iters	2.48 ± 0.09
Mel pseudoinverse	Local Weighted Sums	2.51 ± 0.09

From these results, we conclude that an ideal solution to *either* magnitude or phase estimation can be coupled with a good heuristic for the other to produce high-quality speech. While the ground truth speech is still significantly more natural than that of ideal+heuristic strategies, the MOS for these methods are only 2-9% worse than the ground truth ($p < 0.05$). Of these two problems, we choose to focus on building strategies for the magnitude estimation problem, as it is well-suited to modern deep learning methodology (outlined in Section 4.12).

As a secondary conclusion, we observe that—for our speech data—using LWS for phase estimation from real spectrograms yields significantly higher MOS than using Griffin-Lim. Given that it is faster *and* yields significantly more natural speech, we recommend that all TTS research use LWS as a phase estimation baseline instead of Griffin-Lim. Henceforth, all of our experiments that require phase estimation use LWS.

4.12 Adversarial vocoding

Our goal is to invert a mel spectrogram feature representation into a time domain waveform representation. In the previous section, we demonstrated the potential of the magnitude estimation subproblem for achieving this goal in combination with the LWS phase estimation heuristic. A common heuristic for magnitude estimation is performed by multiplying the mel spectrogram with the approximate inverse of the mel transformation matrix. Since the mel spectrogram is a lossy compression of the magnitude spectrogram, a simple linear transformation is an oversimplification of the magnitude estimation problem.

In order to perform more accurate magnitude estimation, we formulate it as a generative modeling problem and propose a Generative Adversarial Network (GAN) [Goodfellow et al., 2014] based solution. GANs are generative models which seek to learn latent structure in the distribution of data. They do this by mapping samples $\boldsymbol{z}$ from a uniform or gaussian distribution p_Z to samples $\boldsymbol{y}$ from another distribution Y, $G : \boldsymbol{z} \rightarrow \boldsymbol{y}$ [Goodfellow et al., 2014]. For our purpose, we use a variation of GAN called *conditional* GAN [Mirza and Osindero, 2014] to model the conditional probability distribution of magnitude spectrograms given a mel spectrogram. The pix2pix method [Isola et al., 2017] demonstrates that this conditioning information can be a structurally-rich image, extending GANs to learn stochastic mappings from one image domain (spectrogram domain in our case) to another. We adapt the pix2pix approach for our task.

The conditional GAN objective to generate appropriate magnitude spectrograms $\boldsymbol{y}$ given mel spectrograms $\boldsymbol{x}$ is:

$$\begin{aligned}\mathcal{L}_{cGAN}(G,D) =& \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}[\log D(\boldsymbol{x},\boldsymbol{y})] + \\ & \mathbb{E}_{\boldsymbol{x},\boldsymbol{z}}[\log(1 - D(\boldsymbol{x}, G(\boldsymbol{x},\boldsymbol{z}))], \end{aligned} \tag{4.4}$$

where the generator G tries to minimize this objective against an adversary D that tries to maximize it. i.e $G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G,D)$. In such a conditional GAN setting, the generator tries to "fool" the discriminator by generating *realistic* magnitude spectrograms that correspond to the conditioning mel spectrogram. Previous works [Isola et al., 2017, Pascual et al., 2017] have shown that it is beneficial to add a secondary component to the generator loss in order to minimize the L_1 distance between the generated output $G(\boldsymbol{x},\boldsymbol{z})$ and the target $\boldsymbol{y}$. This way, the adversarial component encourages the generator to generate more realistic results, while the L_1 objective ensures the generated output is close to the target.

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}}[||\boldsymbol{y} - G(\boldsymbol{x},\boldsymbol{z})||_1]. \tag{4.5}$$

Our final objective therefore becomes:

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda \mathcal{L}_{L1}(G). \tag{4.6}$$

Here, λ is a hyperparameter which determines the trade-off between the L_1 loss and adversarial loss.

4.12.1 Network architecture

Figure 4.8 shows the high level training setup for adversarial inversion of the mel spectrogram representation into the magnitude spectrogram.

Generator The generator network G takes as input the linear-amplitude mel spectrogram representation x of shape $(n, melBins)$ and generates a magnitude spectrogram of shape $(n, 513)$. The generator first estimates the magnitude spectrorgram through a fixed (non trainable) linear projection of the mel spectrogram using the approximate inverse of the mel transformation matrix. The estimated magnitude spectrogram goes through a convolution based encoder-decoder architecture with skip connections as in pix2pix [Isola et al., 2017]. Past works [Mathieu et al., 2015, Isola et al., 2017] have noted that generators similar to our own empirically learn to ignore latent codes leading to deterministic models. We adopt the same policy of using dropout at both training and test time to force the model to be stochastic (as our task is not a one-to-one mapping). Additionally, we also train a smaller generator *(Advoc - small)* with fewer convolutional layers and fewer convolutional channels. We omit the specifics of our architecture for brevity, however

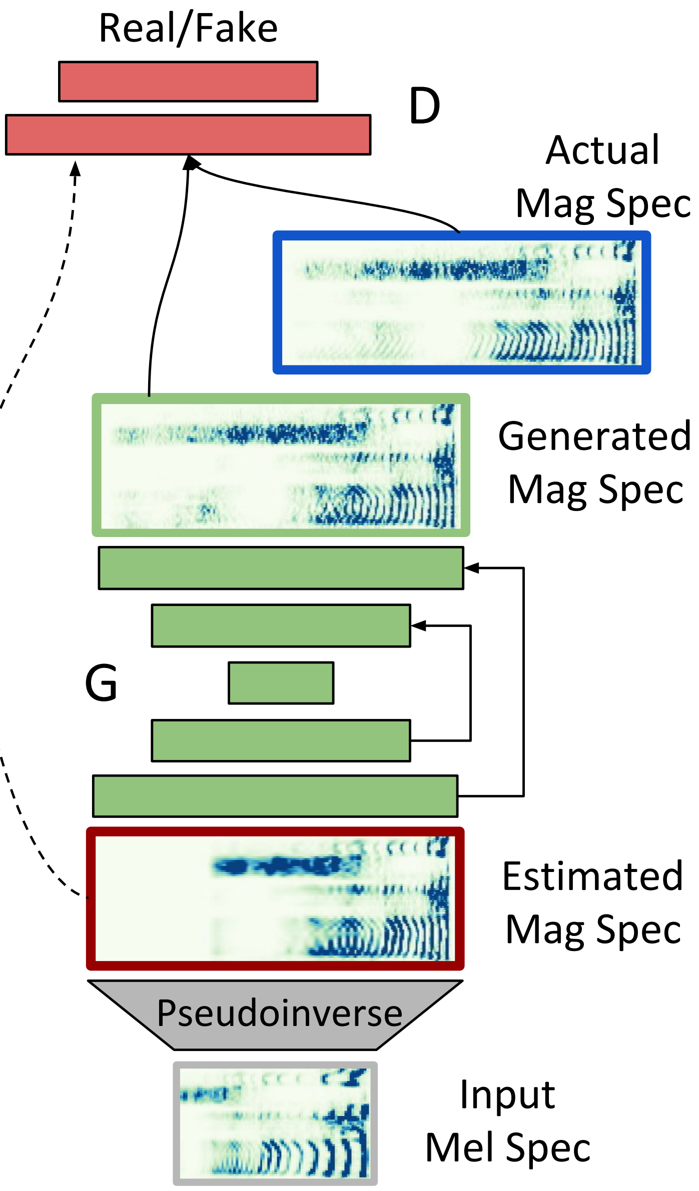


Figure 4.8: Adversarial Vocoder Model: The generator performs an image-to-image translation from the estimated magnitude spectrogram to the actual magnitude spectrogram guided by an adversarial loss from the discriminator and the L_1 distance between the generated and actual magnitude spectrogram

we point to our codebase (link in footnote of previous page) for precise model implementations.

Discriminator Previous works have found that training generators similar to our own using just an L_1 or L_2 loss produces images with reasonable global structure (spatial relationships preserved) but poor local structure (blurry) [Pathak et al., 2016, Zhang et al., 2016]. As in [Isola et al., 2017], we combine an L_1 loss with a discriminator which operates on *patches* (subregions) of a spectrogram to help improve the "sharpness" of the output. Our discriminator takes as input the estimated spectrogram and *either* the generated or real magnitude spectrogram. Thus, in order to satisfy the discriminator, the generator needs to produce magnitude spectrograms that both correspond to the mel spectrogram *and* look realistic.

To complete our adversarial vocoding pipeline, we combine generated magnitude spectrograms with LWS-estimated phase spectrograms and use the inverse STFT to synthesize audio.

4.13 Experiments

We focus our primary empirical study on the publicly available LJ Speech dataset [Ito, 2017], which is popularly used in TTS research [Prenger et al., 2018, Yamamoto., 2018]. The dataset contains 13k short audio clips (24 hours) of a single speaker reading from non-fiction books.

Audio is processed using the feature extraction process described in Section 4.10. We train three models for $\textit{melBins} \in \{20, 40, 80\}$ to study the feasibility of our technique for varying levels of mel compression. Each of the models is trained for 100,000 mini-batch iterations using a batch size of 8 which corresponds to 12 hours of wall clock training time using a NVIDIA 1080Ti GPU. We set the regularization parameter $\lambda = 10$ and use the Adam optimizer [Kingma and Ba, 2015] ($\alpha = 0.0002$).

4.13.1 Vocoding LJ Speech mel spectrograms

In this study we are concerned with vocoding both real mel spectrograms extracted from the LJ Speech dataset *and* mel spectrograms generated by a language-to-spectrogram model [Shen et al., 2018] trained on LJ Speech. We compare both our large (*AdVoc*) and small (*AdVoc-small*) adversarial vocoder models to the mel pseudoinverse magnitude estimation heuristic combined with LWS (*Pseudoinverse*), a *WaveNet* vocoder [Shen et al., 2018], and the recent *WaveGlow* [Prenger et al., 2018] method. We cannot directly compare to the *Parallel WaveNet* approach because it is an end-to-end TTS method rather than a vocoder [van den Oord et al., 2017a].

We randomly select 100 examples from the holdout dataset of LJ Speech and convert them to mel spectrograms. We also synthesize mel spectrograms for each transcript of these same examples using the language-to-spectrogram module from Tacotron 2 [Shen et al., 2018]. We vocode both the real and synthetic spectrograms to audio using the five methods outlined in the previous paragraph. Audio from each method can be found in our sound examples (footnote of first page).

To gauge the relative quality of our methods against others, we conduct two mean opinion score (comparative) studies with human judges on Amazon Mechanical Turk. In the first user study, we show each judge a randomly-ordered batch of six versions of the same utterance: the original utterance and the spectrogram of that utterance vocoded by the five aforementioned methods. In the second user study, we show each judge a batch consisting of the real utterance and five vocodings of a synthetic spectrogram with the same transcript. Judges are asked to rate the naturalness of each on a subjective 1–5 scale with 1 point increments. Each batch is reviewed by 8 different reviewers resulting in 800 evaluations of each strategy. We display mean opinion scores in Table 4.2. We also include the speed of each method (relative to real time) as measured on GPU, and the sizes of each model's parameters in megabytes.

Our results demonstrate that—for both real *and* synthetic spectrograms—our adversarial

Table 4.3: Comparison of vocoding methods on mel spectrograms with 80 bins. We display comparative mean opinion scores from two separate user studies for vocoding spectrograms extracted from real speech (MOS-Real) and spectrograms generated by a state-of-the-art TTS method (MOS-TTS) with 95% confidence intervals. $\times$ RT denotes the speed up over real time; higher is faster. MB denotes the size of each model in megabytes.

Source	MOS-Real	MOS-TTS	$\times$ RT	MB
Real data	4.16 ± 0.06	4.28 ± 0.07	1.000	
Pseudoinverse	2.91 ± 0.10	2.12 ± 0.09	8.836	0.2
WaveNet [van den Oord et al., 2016]	3.98 ± 0.07	3.87 ± 0.07	0.003	95.0
WaveGlow [Prenger et al., 2018]	4.09 ± 0.06	3.89 ± 0.07	1.229	334.7
AdVoc	3.78 ± 0.07	2.91 ± 0.08	3.111	207.7
AdVoc-small	3.68 ± 0.07	3.09 ± 0.07	3.437	16.0

Table 4.4: Comparison of heuristic and adversarial vocoding of spectrograms with different levels of mel compression. Adversarial vocoding can vocode highly compressed mel spectrograms with relatively less drop in speech naturalness as compared to a heuristic.

Source	*melBins*	MOS
Real data		4.05 ± 0.07
Pseudoinverse	20	2.68 ± 0.10
Pseudoinverse	40	2.84 ± 0.10
Pseudoinverse	80	3.25 ± 0.09
AdVoc	20	3.75 ± 0.07
AdVoc	40	3.79 ± 0.07
AdVoc	80	3.86 ± 0.07

magnitude estimation technique (AdVoc) significantly outperforms magnitude estimation using the psuedoinverse of the mel basis. Our method is more than 1000$\times$ faster than the autoregressive WaveNet vocoder and 2.5$\times$ faster than WaveGlow vocoder.

Additionally, we train our models to perform magnitude estimation on representations with higher compression. Specifically, we train our model to vocode mel spectrograms with 20, 40 and 80 bins. We compare our adversarial magnitude estimation method against magnitude estimation using the pseudoinverse of the mel basis. We conduct a comparative user study using the same methodology as previously outlined. Our results in Table 4.4 demonstrate that our model can vocode highly compressed mel spectrogram representations with relatively little drop in the

Table 4.5: Combining our adversarial vocoding approach with GAN-generated mel spectrograms outperforms our prior work in unsupervised generation of individual words by all metrics.

	Quantitative	*Qualitative*	
Source	Inception score	Acc.	MOS
Real data	8.01 ± 0.24	0.95	3.9 ± 0.15
WaveGAN [Donahue et al., 2019a]	4.67 ± 0.01	0.58	2.3 ± 0.18
SpecGAN [Donahue et al., 2019a] + Griffin-Lim	6.03 ± 0.04	0.66	1.9 ± 0.17
MelSpecGAN + AdVoc	6.63 ± 0.03	0.71	3.4 ± 0.20

perceived audio quality as compared to the psuedoinversion baseline (audio examples in footnote of first page).

4.13.2 Unsupervised audio synthesis

In this section we are concerned with the *unsupervised* generation of speech (as opposed to supervised generation in the case of TTS). We focus on the SC09 digit generation task proposed in our previous work [Donahue et al., 2019a], where the goal is to learn to generate examples of spoken digits "zero" through "nine" *without* labels. We first train a GAN to generate mel spectrograms of spoken digits (*MelSpecGAN*), then train an adversarial vocoder to generate audio conditioned on those spectrograms. Using a pretrained digit classifier, we calculate an Inception score [Salimans et al., 2016] for our approach, finding it to outperform our previous state-of-the-art results by 9%. We also calculate an "accuracy" by comparing human labelings to classifier labels for our generated digits, and also solicit subjective speech quality ratings from listeners, finding that our adversarial vocoding-based method outperforms our previous results (Table 4.5).

4.14 Conclusion

We presented WaveGAN, the first application of GANs to unsupervised audio generation. WaveGAN is fully parallelizable and can generate hours of audio in only a few seconds. In its current form, WaveGAN can be used for creative sound design in multimedia production. In our future work we plan to extend WaveGAN to operate on variable-length audio and also explore a variety of label conditioning strategies. By providing a template for modifying image generation models to operate on audio, we hope that this work catalyzes future investigation of GANs for audio synthesis.

We have additionally shown that we can improve upon the results of WaveGAN by instead generating perceptually-informed spectrograms and training a GAN to help vocode these spectrograms to audio. We first demonstrated that solutions to *either* the magnitude or phase estimation subproblems within common vocoding pipelines can result in high-quality speech. Then we demonstrated a learning-based method for magnitude estimation which significantly improves upon popular heuristics for this task. In addition to representing the state of the art in unsupervised synthesis of small-vocabulary speech, we demonstrate that our method can integrate with an existing TTS pipeline to provide comparatively fast waveform synthesis.

4.15 Acknowledgements

I would like to thank Bo Li for helpful discussions about this work, and also Peter Boesman and Colin Raffel for providing training data. This research was supported by the UC San Diego Chancellors Research Excellence Scholarship program, the Unity Global Graduate Fellowship program, and the UC San Diego Department of Computer Science. GPUs used for this work were provided by the HPC @ UC program and donations from NVIDIA.

Chapter 4 contains material found in the following two papers. (1) *Adversarial audio synthesis*. 2018. McAuley, Julian; Puckette, Miller [Donahue et al., 2019a]. International Confer-

ence on Learning Representations 2019. (2) *Expediting TTS synthesis with adversarial vocoding*. 2019. Neekhara, Paarth; Puckette, Miller; Dubnov, Shlomo; McAuley, Julian [Neekhara et al., 2019]. Currently under review for publication. The dissertation/thesis author was the primary investigator and author of these papers.

Part III

New musical interactions

In Part II, I discussed machine learning strategies for building generative models of music at both the symbolic and acoustic level. Here I discuss how to use these and other machine learning models for music to enable new musical interactions. In Chapter 5, I present *Piano Genie*, a system which grants a degree of piano improvisation literacy to non-musicians. Piano Genie pairs the ideas from Chapter 3 with a simple and intuitive interface allowing non-musicians to improvise music in real time. I also describe *Dance Dance Convolution*, a system which can benefit musicians and non-musicians alike by automatically generating choreographies for a music-based video game called Dance Dance Revolution (Chapter 6). Dance Dance Convolution enhances the experience of players of the game by allowing them to dance to their *personal* music library. Finally, I discuss other interfaces I have built that use generative models to enable interactive synthesis of music and other types of multimedia (Chapter 7).

Chapter 5

Piano Genie

5.1 Introduction

While most people have an innate sense of and appreciation for music, comparatively few are able to participate meaningfully in its creation. A non-musician could endeavor to achieve proficiency on an instrument, but the time and financial requirements may be prohibitive. Alternatively, a non-musician could operate a system which automatically generates complete songs at the push of a button, but this would remove any sense of ownership over the result. We seek to sidestep these obstacles by designing an intelligent interface which takes high-level specifications provided by a human and maps them to plausible musical performances.

The practice of "air guitar" offers hope that non-musicians can provide such specifications [Godøy et al., 2005]—performers strum fictitious strings with rhythmical coherence and even move their hands up and down an imaginary fretboard in correspondence with *melodic contours*, i.e. rising and falling movement in the melody. This suggests a pair of attributes which may function as an effective communication protocol between non-musicians and generative music systems: 1) rhythm, and 2) melodic contours. In addition to air guitar, rhythm games such as *Guitar Hero* [Harmonix, 2005] also make use of these two attributes. However, both

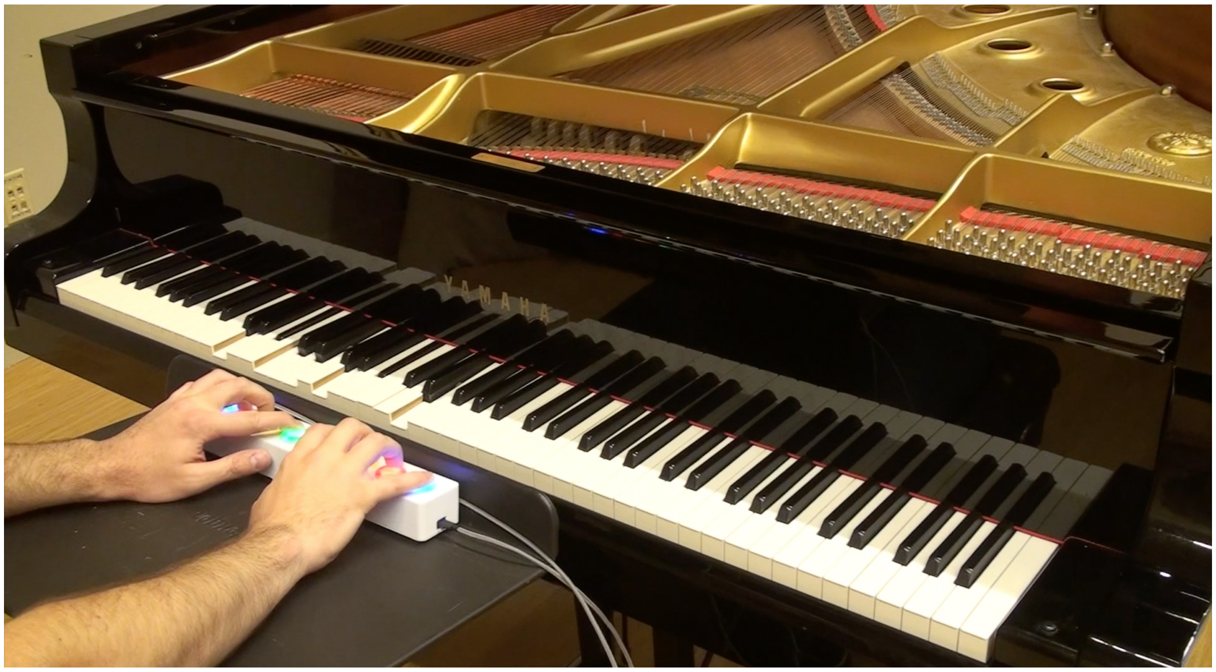


Figure 5.1: Using Piano Genie to improvise on a Disklavier (motorized piano) via MIDI.

experiences only allow for the imitation of experts and provide no mechanism for the *creation* of music.

In this work, we present *Piano Genie*, an intelligent controller allowing non-musicians to improvise on the piano while retaining ownership over the result (Figure 5.1). In our web demo, a participant improvises on eight buttons, and their input is translated into a piano performance by a neural network running in the browser in real-time.[1] Piano Genie has similar performance mechanics to those of a real piano: pressing a button will trigger a note that sounds until the button is released. Multiple buttons can be pressed simultaneously to achieve polyphony. The mapping between buttons and pitch is non-deterministic, but the performer can control the overall form by pressing higher buttons to play higher notes and lower buttons to player lower notes.

Because we lack examples of people performing on "miniature pianos", we adopt an unsupervised strategy for learning the mappings. Specifically, we use the *autoencoder* setup,

[1]**Web Demo**: https://piano-genie.glitch.me, **Video**: https://youtu.be/YRb0XAnUpIk,
Training Code: https://bit.ly/2Vv78Gx, **Inference Code**: https://bit.ly/2QolPrb

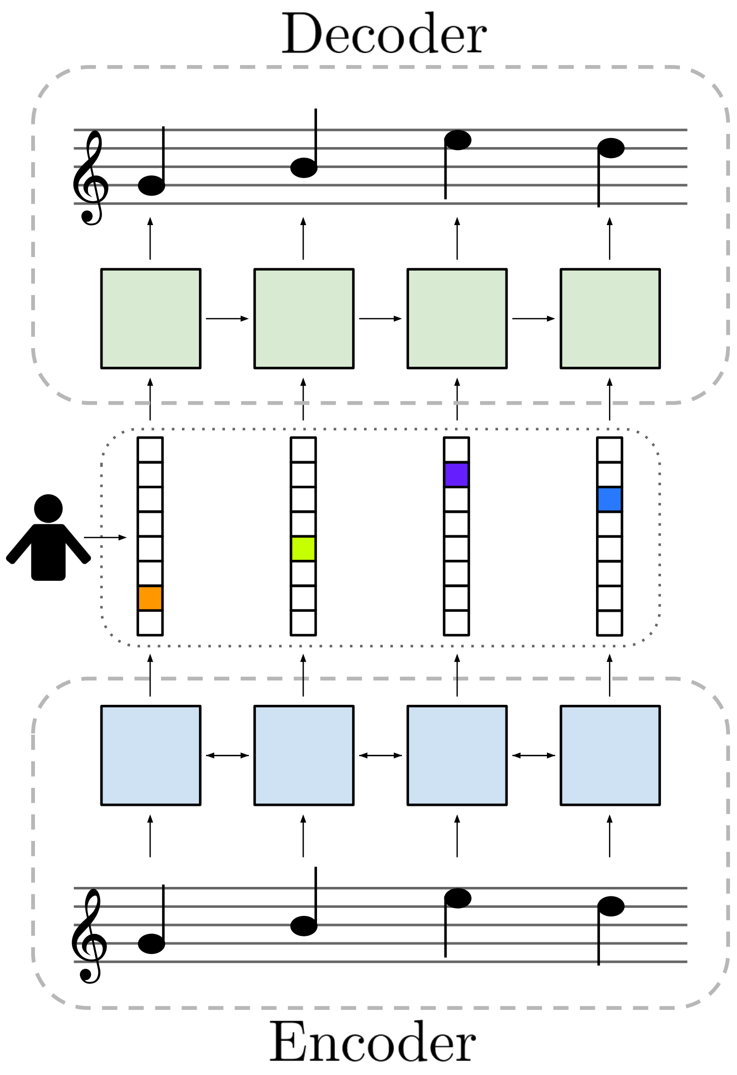


Figure 5.2: Piano Genie consists of a discrete sequential autoencoder. A bidirectional RNN encodes discrete piano sequences (88 keys) into smaller discrete latent variables (8 "buttons"). The unidirectional decoder is trained to map the latents back to piano sequences. During inference, the encoder is replaced by a human improvising on buttons.

where an *encoder* learns to map 88-key piano sequences to 8-button sequences, and a *decoder* learns to map the button sequences back to piano music (Figure 5.2). The system is trained end-to-end to minimize reconstruction error. At performance time, we replace the encoder's output with a user's button presses, evaluating the decoder in real time.

5.2 Related Work

Perception of melodic contour is a skill acquired in infancy [Trehub et al., 1984]. This perception is important for musical memory and is somewhat invariant to transposition and changes in intervals [Dowling, 1978, Huron, 1996]. The act of *sound tracing*—moving one's hands in the air while listening to music—has been studied in music information retrieval [Parsons, 1975, Godøy and Jensenius, 2009, Nymoen et al., 2011, Kelkar and Jensenius, 2017, Lartilot, 2018]. It has been suggested that the relationship between sound tracings and pitch is non-linear [Eitan et al., 2014, Kelkar et al., 2018]. Like Piano Genie, some systems use user-provided contours to *compose* music [Roy et al., 2014, Kitahara et al., 2017], though these systems generate complete songs rather than allowing for real-time improvisation. An early game by Harmonix called *The Axe* [Harmonix, 1998] allowed users to improvise in real time by manipulating contours which indexed pre-programmed melodies.

There is extensive prior work [Lee et al., 1992, Bevilacqua et al., 2005, Fiebrink et al., 2009, Gillian and Knapp, 2011] on *supervised* learning of mappings from different control modalities to musical gestures. These approaches require users to provide a training set of control gestures and associated labels. There has been less work on *unsupervised* approaches, where gestures are automatically extracted from arbitrary performances. Scurto and Fiebrink [Scurto and Fiebrink, 2016] describe an approach to a "grab-and-play" paradigm, where gestures are extracted from a performance on an arbitrary control surface, and mapped to inputs for another. Our approach differs in that the controller is fixed and integrated into our training methodology, and we require no example performances on the controller.

5.3 Methods

We wish to learn a mapping from sequences $\boldsymbol{y} \in [0, 8)^n$, i.e. amateur performances of n presses on eight buttons, to sequences $\boldsymbol{x} \in [0, 88)^n$, i.e. professional performances on an 88-key

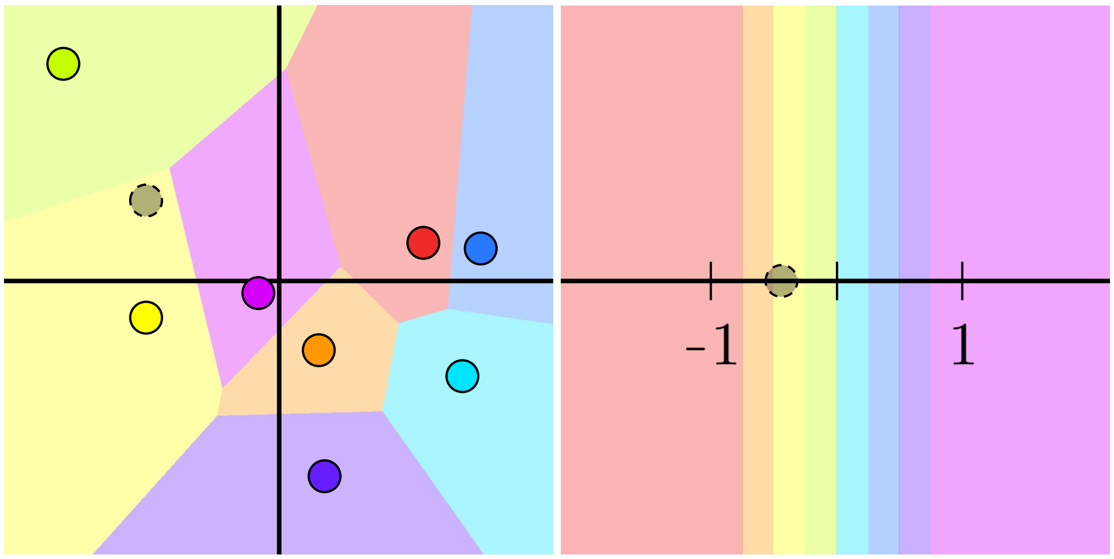


Figure 5.3: Comparison of the quantization scheme for two autoencoder strategies with discrete latent spaces. The VQ-VAE (left) learns the positions of k centroids in a d-dimensional embedding space (in this figure $k = 8, d = 2$, in our experiments $k = 8, d = 4$). An encoder output vector (grey circle) is quantized to its nearest centroid (yellow circle) before decoding. Our IQAE strategy (right) quantizes a scalar encoder output (grey) to its nearest neighbor (yellow) among $k = 8$ centroids evenly spaced between -1 and 1.

piano. To preserve a one-to-one mapping between buttons pressed and notes played, we assume that both $\boldsymbol{y}$ and $\boldsymbol{x}$ are monophonic sequences.[2] Given that we lack examples of $\boldsymbol{y}$, we propose using the autoencoder framework on examples $\boldsymbol{x}$. Specifically, we learn a deterministic mapping $\text{enc}(\boldsymbol{x}) : [0, 88)^n \mapsto [0, 8)^n$, and a stochastic inverse mapping $P_{\text{dec}}(\boldsymbol{x}|\text{enc}(\boldsymbol{x}))$.

We use LSTM recurrent neural networks (RNNs) [Hochreiter and Schmidhuber, 1997] for both the encoder and the decoder. For each input piano note, the encoder outputs a real-valued scalar, forming a sequence $\text{enc}_s(\boldsymbol{x}) \in \mathbb{R}^n$. To discretize this into $\text{enc}(\boldsymbol{x})$ we quantize it to $k = 8$ buckets equally spaced between -1 and 1 (Figure 5.3), and use the straight-through estimator [Bengio et al., 2013] to bypass this non-differentiable operation in the backwards pass. We refer to this contribution as the integer-quantized autoencoder (IQAE); it is inspired by two papers from the image compression literature that also use autoencoders with discrete

[2]This does not prevent our method from working on polyphonic piano music; we just consider each key press to be a separate event.

bottlenecks [Ballé et al., 2017, Theis et al., 2017]. We train this system end-to-end to minimize:

$$L = L_{\text{recons}} + L_{\text{margin}} + L_{\text{contour}} \tag{5.1}$$

$$\begin{aligned} L_{\text{recons}} &= -\Sigma \log P_{\text{dec}}(\boldsymbol{x}|\text{enc}(\boldsymbol{x})) \\ L_{\text{margin}} &= \Sigma \max(|\text{enc}_s(\boldsymbol{x})| - 1, 0)^2 \\ L_{\text{contour}} &= \Sigma \max(1 - \Delta\boldsymbol{x}\Delta\text{enc}_s(\boldsymbol{x}), 0)^2 \end{aligned}$$

Together, L_{recons} and L_{margin} constitute our proposed IQAE objective. The former term minimizes reconstruction loss of the decoder (as is typical of autoencoders). To agree with our discretization strategy, the latter term discourages the encoder from producing values outside of $[-1,1]$. We also contribute a musically motivated regularization strategy which gives the model an awareness of melodic contour. By comparing the finite differences (musical intervals in semitones) of the input $\Delta\boldsymbol{x}$ to the finite differences of the real-valued encoder output $\Delta\text{enc}_s(\boldsymbol{x})$, the L_{contour} term encourages the encoder to produce "button contours" that match the shape of the input melodic contours.

5.4 Experiments and Analysis

We train our model on the Piano-e-Competition data [Hawthorne et al., 2019], which contains around 1400 performances by skilled pianists. We flatten each polyphonic performance into a single sequence of notes ordered by start time, breaking ties by listing the notes of a chord in ascending pitch order. We split the data into training, validation and testing subsets using an 8 : 1 : 1 ratio. To keep the latency low at inference time, we use relatively small RNNs consisting of two layers with 128 units each. We use a bidirectional RNN for the encoder, and a unidirectional RNN for the decoder since it will be evaluated in real time. Our training examples

Table 5.1: Quantitative results comparing an RNN language model, the VQ-VAE ($k = 8, d = 4$), and our proposed IQAE model ($k = 8$) with and without contour regularization. ΔT adds time shift features to model. PPL is perplexity: $e^{L_{\text{recons}}}$. CVR is contour violation ratio: the proportion of timesteps where the sign of the melodic interval $\neq$ that of the button interval. Gold is the mean squared error in button space between the encoder outputs for familiar melodies and manually-created gold standard button sequences for those melodies. Lower is better for all metrics.

Configuration	PPL	CVR	Gold
Language model	15.44		
$+\Delta T$	11.13		
VQ-VAE [van den Oord et al., 2017b]	3.31	.360	9.69
$+\Delta T$	2.82	.465	9.15
IQAE	3.60	.371	5.90
$+L_{\text{contour}}$	3.53	.002	1.70
$+L_{\text{contour}} + \Delta T$	3.16	.004	1.61

consist of 128-note subsequences randomly transposed between $[-6, 6)$ semitones. We perform early stopping based on the reconstruction error on the validation set.

As a baseline, we consider an LSTM "language model"—equivalent to the decoder portion of our IQAE without button inputs—trained to simply predict the next note given previous notes. This is a challenging sequence modeling task, among other reasons because the monophonic sequences will frequently jump between the left and the right hand. To allow the network to factor in timing into its predictions, we add in a ΔT feature to the input, representing the amount of time since the previous note quantized into 32 buckets evenly spaced between 0 and 1 second. This language model baseline is not unlike our previous work on Performance RNN [Simon and Oore, 2017], though in that work our goal was to predict not only notes but also timing information and dynamics (here, timing is provided and dynamics are ignored).

We also compare to the VQ-VAE strategy [van den Oord et al., 2017b], an existing discrete autoencoder approach. The VQ-VAE strategy discretizes based on proximity to learned centroids within an embedding vector space (Figure 5.3) as opposed to the fixed scalar centroids in our IQAE (Figure 5.3). Accordingly, it is not possible to apply the same contour regularization strategy to the VQ-VAE, and the meaning of the mapping between the buttons and the resultant notes is less interpretable.

5.4.1 Analysis

To evaluate our models, we calculate two metrics on the test set: 1) the perplexity (PPL) of the model $e^{L_{\text{recons}}}$, and 2) the ratio of contour violations (CVR), i.e. the proportion of timesteps where the sign of the button interval disagrees with the sign of the note interval. We also manually create "gold standard" button sequences for eight familiar melodies (e.g. *Frère Jacques*), and measure the mean squared error in button space between these gold standard button sequences and the output of the encoder for those melodies (Gold). We report these metrics for all models in Table 5.1.

As expected, all of the autoencoder models outperformed the language model in terms of reconstruction perplexity. The VQ-VAE models achieved better reconstruction costs than their IQAE counterparts, but produced non-intuitive button sequences as measured by comparison to gold standards. In Figure 5.4, we show a qualitative comparison between the button sequences learned for a particular input by the VQ-VAE and our IQAE with contour regularization. The sequences learned by our contour-regularized IQAE model are visually more similar to the input.

Interestingly, the IQAE model regularized with the L_{contour} penalty had better reconstruction than its unregularized counterpart. It is possible that the contour penalty is making the decoder's job easier by limiting the space of mappings that the encoder can learn. The L_{contour} penalty was effective at aligning the button contours with melodic contours; the encoder violates the melodic contour at less than 1% of timesteps. The ΔT features improved reconstruction for all models.

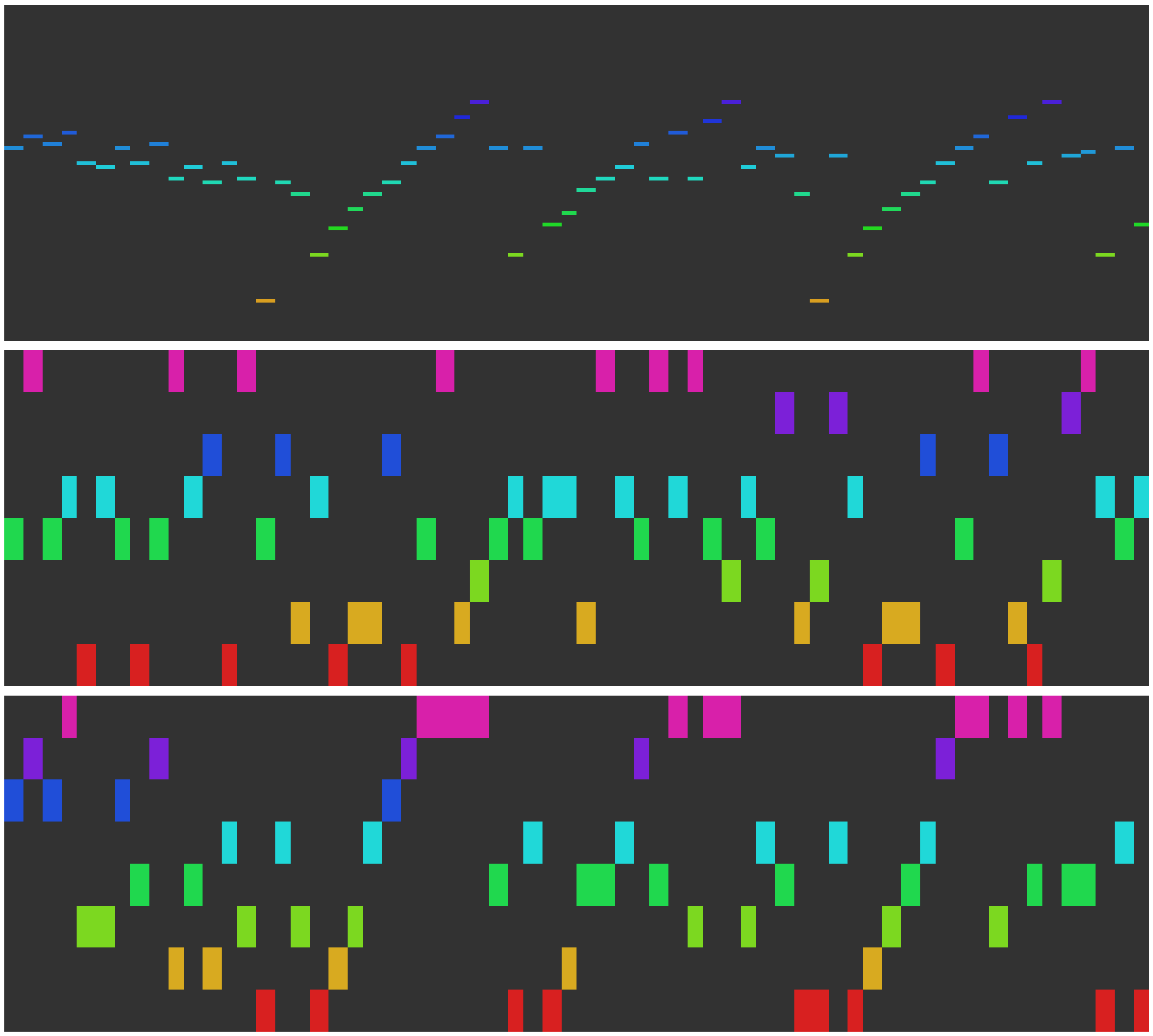


Figure 5.4: Qualitative comparison of the 8-button encodings for a given melody (top) by the VQ-VAE (middle) and our IQAE with L_{contour} (bottom). Horizontal is note index. The encoding learned by the IQAE echoes the contour of the musical input.

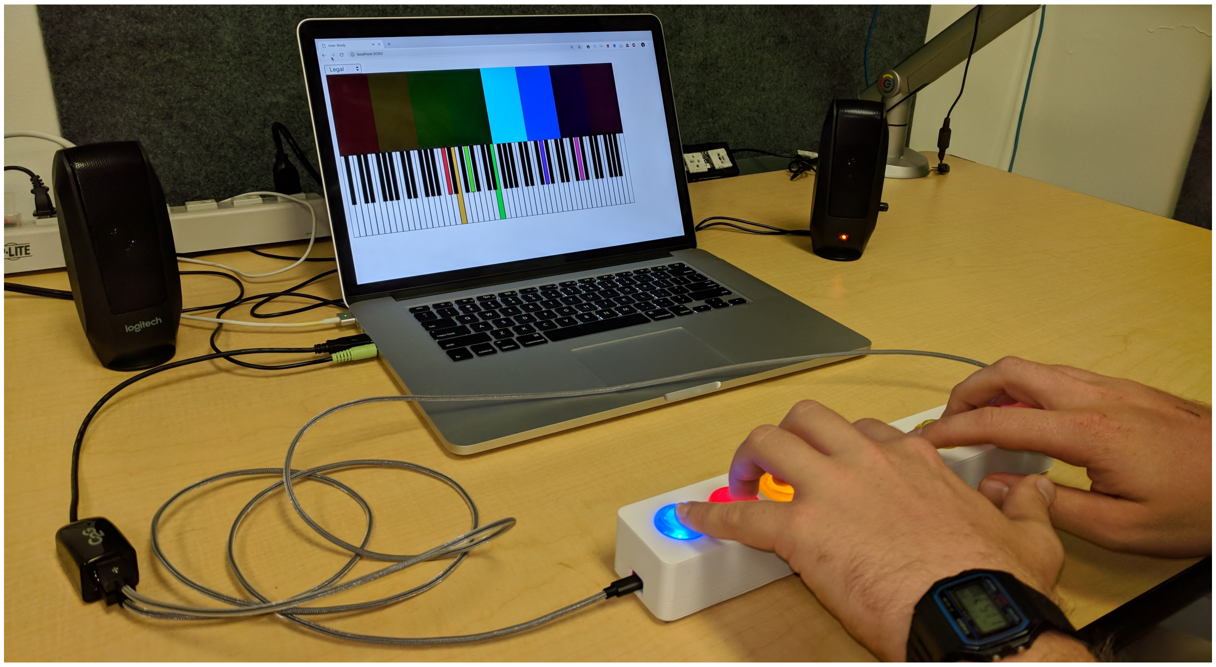


Figure 5.5: A user engages with the Piano Genie web interface during our user study.

Table 5.2: Results for a small ($n = 8$) user study for Piano Genie. Participants were given up to three minutes to improvise on three possible mappings between eight buttons and an 88-key piano: 1) (*G-maj*) the eight buttons are mapped to a G-major scale, 2 (*language model*) all buttons trigger our baseline language model, 3 (*Piano Genie*) our proposed method. (*Time*) is the average amount of time in seconds that users improvised with a mapping. (*Per.*, *Mus.*, *Con.*) are the respective averages users expressed for enjoyment of performance experience, enjoyment of music, and level of control.

Mapping	Time (s)	Per.	Mus.	Con.
G-maj scale	92.6	3.125	3.250	4.625
Language model	127.9	3.750	3.125	1.750
Piano Genie	144.1	4.375	3.125	3.125

5.5 User Study

While our above analysis is useful as a sanity check, it offers limited intuition about how Piano Genie behaves in the hands of users. Accordingly, we designed a user study to compare three mappings between eight buttons and a piano:

1. (*G-maj*) the eight buttons are deterministically mapped to a G major scale
2. (*language model*) pressing any button triggers a prediction by our baseline musical language model
3. (*Piano Genie*) our IQAE model with contour regularization

Our reason for including the G-maj baseline is to gauge the performance experience of a mapping where a user has deterministic control but has to depend upon their own musical knowledge to produce patterns of interest. Our motivation for comparing Piano Genie to the language model is to determine how the performance experience is altered by including melodic control control in addition to rhythm.

Eight participants were given up to three minutes to improvise with each mapping (Figure 5.5). The length of time they spent on each mapping was recorded as an implicit feedback signal. Participants reported a wide range of experience with piano performance: three had "no experience", half had "some experience", and one had "substantial experience". After each mapping, participants were asked to what extent they agreed with the following statements:

A. "I enjoyed the experience of performing this instrument"

B. "I enjoyed the music that was produced while I played"

C. "I was able to control the music that was produced"

This survey was conducted on a five-level Likert scale [Likert, 1932] and we convert the responses to a 1-5 numerical scale in order to compare averages (Table 5.2).

When asked about their enjoyment of the performance experience, all eight users preferred Piano Genie to G-maj, while seven preferred Piano Genie to the language model. Five out of eight users enjoyed the music produced by Piano Genie more than that produced by G-maj. As expected, no participants said that Piano Genie gave them more control than the G-maj scale. However, all eight said that Piano Genie gave them more control than the language model.

Though our user study was too limited in scope to make meaningful statistical claims, informally speaking the participants were quite enthusiastic about Piano Genie in comparison to the other mappings. One participant said "there were some times when [Piano Genie] felt like it was reading my mind". Another participant said "how you can cover the entire keyboard with only 8 buttons is pretty cool." One mentioned that the generative component helped them overcome stage fright; they could blame Piano Genie for perceived errors and take credit for perceived successes. Several participants cited their inability to produce the same notes when playing the same button pattern as a potential drawback; enabling these patterns of repetition is a promising avenue for future work. The participants with less piano experience said they would have liked some more instruction about types of gestures to perform.

5.6 Web demo details

We built a web demo (polished demo: https://piano-genie.glitch.me; earlier version with all models from this paper: https://bit.ly/2FaMeI4) for Piano Genie to allow us to both improvise with our models and conduct our user study. Our web demo uses TENSORFLOW.JS[3] to run our separately-trained neural networks in the browser in real-time. When a user presses a button, we pass this value into our trained decoder and run a forward pass producing a vector of 88 logits representing the piano keys. We divide the logits by a *temperature* parameter before normalizing them to a probability distribution with a *softmax*. If the temperature is 0, sampling from this

[3]https://js.tensorflow.org

distribution is equivalent to *argmax*. Informally, we found a temperature of 0.25 to yield a satisfying experience.

For the models that use the ΔT features, we have to wait until the user presses a key to run a forward pass of the neural network. For the models that do not use these features, we can run the computation for all 8 possible buttons in advance. This allows us to both reduce the latency and display a helpful visualization of the possible model outputs contingent upon the user pressing any of the buttons (only available in our earlier demo https://bit.ly/2FaMeI4).

To build an interface for Piano Genie that would be more inviting than a computer keyboard, we 3D-printed enclosures for eight arcade buttons which communicate with the computer via USB (Figure 5.1).[4] Due to technical limitations of our USB microcontroller, we ended up building two boxes with four buttons instead of one with eight. This resulted in multiple unintended but interesting control modalities. Several users rearranged the boxes from a straight line to different 2D configurations. Another user—a flutist—picked up the controllers and held them to their mouth. A pair of users each took a box and improvised a duet.

5.7 Conclusion

We have proposed Piano Genie, an intelligent controller which grants non-musicians a degree of piano improvisation literacy. Piano Genie has an immediacy not shared by other work in this space; sound is produced the moment a player interacts with our system rather than requiring laborious configuration. Additionally, the player is kept in the improvisational loop as they respond to the generative procedure in real-time. We believe that the autoencoder framework is a promising approach for learning mappings between complex interfaces and simpler ones, and hope that this work encourages future investigation of this space.

[4]https://learn.adafruit.com/arcade-button-control-box/overview

5.8 Acknowledgements

Special thanks to Monica Dinculescu for developing the polished web demo for Piano Genie. Thanks to Adam Roberts, Anna Huang, Ashish Vaswani, Ben Poole, Colin Raffel, Curtis Hawthorne, Doug Eck, Jesse Engel, Jon Gillick, Yingtao Tian and others at Google AI for helpful discussions and feedback throughout this work. I would also like to thank Julian McAuley, Stephen Merity, and Tejaswinee Kelkar for helpful conversations. Finally, I would like to thank all participants of our user study.

Chapter 5, in full, is a reprint of the material as it appears in *Piano Genie*. 2019. Simon, Ian; Dieleman, Sander [Donahue et al., 2019b]. ACM conference on Intelligent User Interfaces 2019. The dissertation/thesis author was the primary investigator and author of this paper.

Chapter 6

Dance Dance Convolution

6.1 Introduction

Dance Dance Revolution (DDR) is a rhythm-based video game with millions of players worldwide [Hoysniemi, 2006]. Players perform steps atop a dance platform, following prompts from an on-screen *step chart* to step on the platform's buttons at specific, musically salient points in time. A player's score depends upon both hitting the correct buttons and hitting them at the correct time. Step charts vary in difficulty with harder charts containing more steps and more complex sequences. The dance pad contains *up*, *down*, *left*, and *right* arrows, each of which can be in one of four states: *on*, *off*, *hold*, or *release*. Because the four arrows can be activated or released independently, there are 256 possible step combinations at any instant.

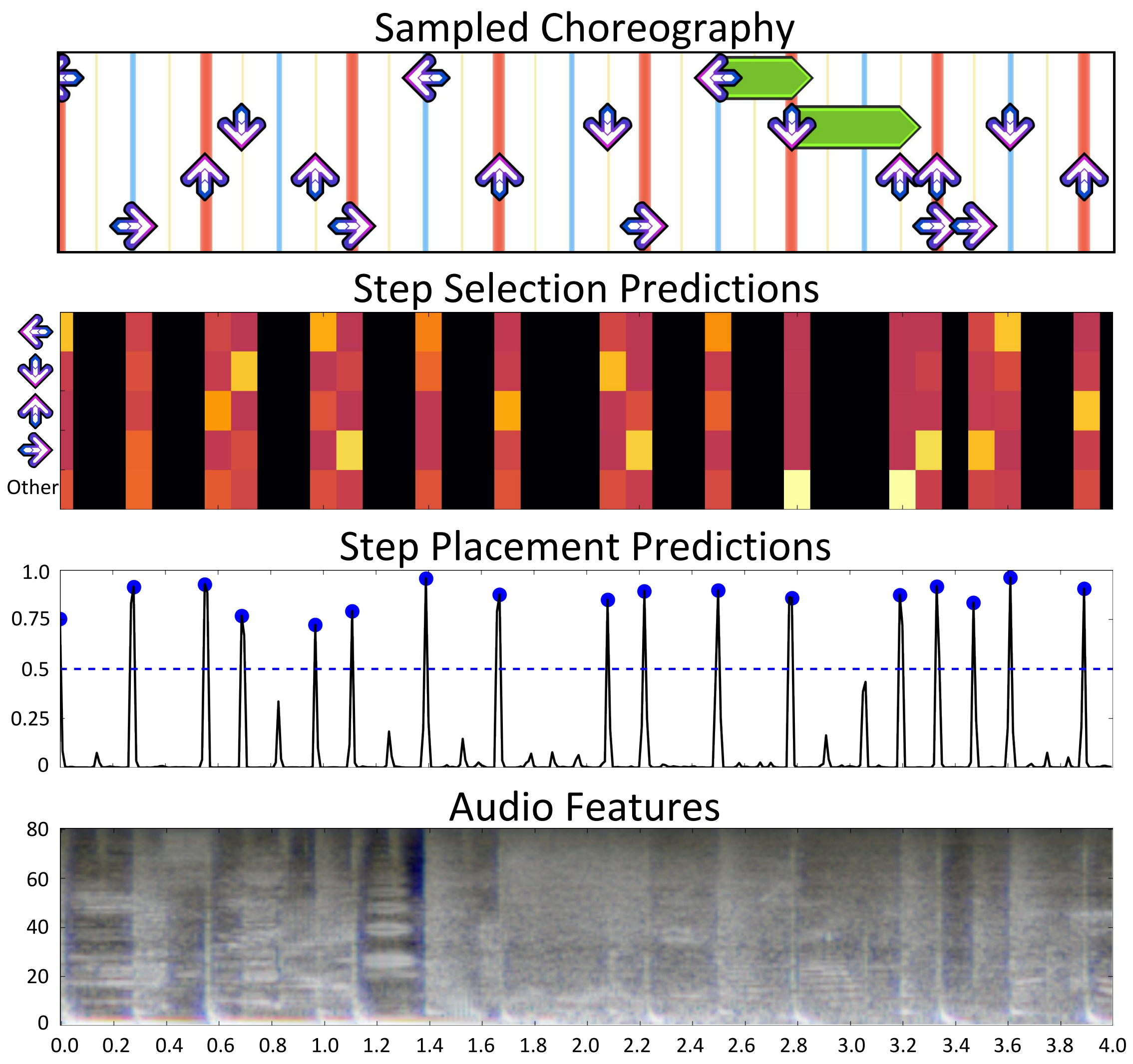


Figure 6.1: Proposed *learning to choreograph* pipeline for four seconds of the song *Knife Party feat. Mistajam - Sleaze*. The pipeline ingests audio features (**Bottom**) and produces a playable DDR choreography (**Top**) corresponding to the audio.

Step charts exhibit rich structure and complex semantics to ensure that step sequences are both challenging and enjoyable. Charts tend to mirror musical structure: particular sequences of steps correspond to different motifs (Figure 6.2), and entire passages may reappear as sections of the song are repeated. Moreover, chart authors strive to avoid patterns that would compel a player to face away from the screen.

The DDR community uses simulators, such as the open-source *StepMania*, that allow fans to create and play their own charts. A number of prolific authors produce and disseminate *packs* of charts, bundling metadata with relevant recordings. Typically, for each song, packs contain one chart for each of five difficulty levels.

Despite the game's popularity, players have some reasonable complaints: For one, packs are limited to songs with favorable licenses, meaning players may be unable to dance to their favorite songs. Even when charts are available, players may tire of repeatedly performing the same charts. Although players can produce their own charts, the process is painstaking and requires significant expertise.

In this paper, we seek to automate the process of step chart generation so that players can dance to a wider variety of charts on any song of their choosing. We introduce the task of *learning to choreograph*, in which we learn to generate step charts from raw audio. Although this task has previously been approached via ad-hoc methods, we are the first to cast it as a learning task in which we seek to mimic the semantics of human-generated charts. We break the problem into two subtasks: First, *step placement* consists of identifying a set of timestamps in the song at which to place steps. This process can be conditioned on a player-specified difficulty level. Second, *step selection* consists of choosing which steps to place at each timestamp. Running these two steps in sequence yields a playable *step chart*. This process is depicted in Figure 6.1.

Progress on learning to choreograph may also lead to advances in music information retrieval (MIR). Our step placement task, for example, closely resembles *onset detection*, a well-studied MIR problem. The goal of onset detection is to identify the times of all musically salient

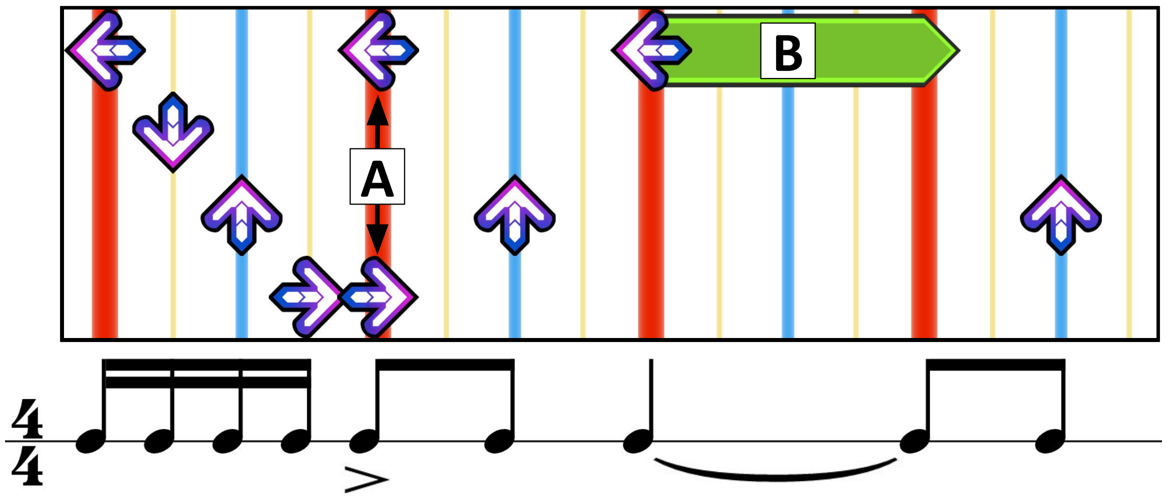


Figure 6.2: A four-beat measure of a typical chart and its rhythm depicted in musical notation. **Red:** quarter notes, **Blue:** eighth notes, **Yellow:** sixteenth notes, **(A)**: *jump* step, **(B)**: *freeze* step

events, such as melody notes or drum strikes. While not every onset in our data corresponds to a DDR step, every DDR step corresponds to an onset. In addition to marking steps, DDR packs specify a metronome click track for each song. For songs with changing tempos, the exact location of each change and the new tempo are annotated. This click data could help to spur algorithmic innovation for *beat tracking* and *tempo detection*.

Unfortunately, MIR research is stymied by the difficulty of accessing large, well-annotated datasets. Songs are often subject to copyright issues, and thus must be gathered by each researcher independently. Collating audio with separately-distributed metadata is nontrivial and error-prone owing to the multiple available versions of many songs. Researchers often must manually align their version of a song to the metadata. In contrast, our dataset is publicly available, standardized and contains meticulously-annotated labels as well as the relevant recordings.

We believe that DDR charts represent an abundant and under-recognized source of annotated data for MIR research. *StepMania Online*, a popular repository of DDR data, distributes over 350*Gb* of packs with annotations for more than 100*k* songs. In addition to introducing a novel task and methodology, we contribute two large public datasets, which we consider to be of notably high quality and consistency.[1] Each dataset is a collection of recordings and step charts. One contains charts by a single author and the other by multiple authors.

For both prediction stages of learning to choreograph, we demonstrate the superior performance of neural networks over strong alternatives. Our best model for step placement jointly learns a convolutional neural network (CNN) representation and a recurrent neural network (RNN), which integrates information across consecutive time slices. This method outperforms CNNs alone, multilayer perceptrons (MLPs), and linear models.

Our best-performing system for step selection consists of a conditional LSTM generative model. As auxiliary information, the model takes *beat phase*, a number representing the fraction of a beat at which a step occurs. Additionally, the best models receive the time difference (measured in beats) since the last and until the next step. This model selects steps that are more consistent with expert authors than the best *n*-gram and fixed-window models, as measured by perplexity and per-token accuracy.

6.1.1 Contributions

In short, our paper offers the following contributions:

- We define *learning to choreograph*, a new task with real-world usefulness and strong connections to fundamental problems in MIR.
- We introduce two large, curated datasets for benchmarking DDR choreography algorithms. They represent an under-recognized source of music annotations.
- We introduce an effective pipeline for learning to choreograph with deep neural networks.[2]

[1]https://github.com/chrisdonahue/ddc

[2]Demonstration showing human choreography alongside our method: https://youtu.be/yUc3O237p9M

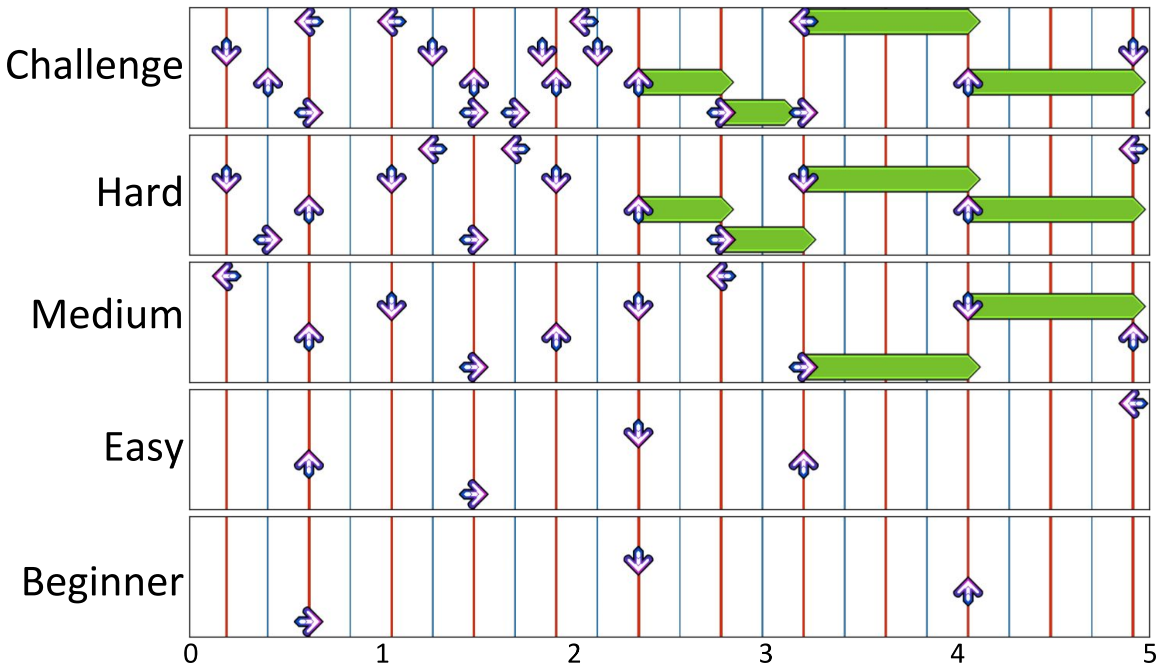


Figure 6.3: Five seconds of choreography by difficulty level for the song *KOAN Sound - The Edge* from the Fraxtil training set.

6.2 Data

Basic statistics of our two datasets are shown in Table 6.1. The first dataset contains 90 songs choreographed by a single prolific author who works under the name *Fraxtil*. This dataset contains five charts per song corresponding to increasing difficulty levels. We find that while these charts overlap significantly, the lower difficulty charts are not strict subsets of the higher difficulty charts (Figure 6.3). The second dataset is a larger, multi-author collection called *In The Groove* (ITG); this dataset contains 133 songs with one chart per difficulty, except for 13 songs that lack charts for the highest difficulty. Both datasets contain electronic music with constant tempo and a strong beat, characteristic of music favored by the DDR community.

Note that while the total number of songs is relatively small, when considering all charts across all songs the datasets contain around 35 hours of annotations and 350,000 steps. The two datasets have similar vocabulary sizes (81 and 88 distinct step combinations, respectively). Around 84% of the steps in both datasets consist of a single, instantaneous arrow.

Table 6.1: Dataset statistics

Dataset	**Fraxtil**	**ITG**
Num authors	1	8
Num packs	3	2
Num songs	90 (3.1 hrs)	133 (3.9 hrs)
Num charts	450 (15.3 hrs)	652 (19.0 hrs)
Steps/sec	3.135	2.584
Vocab size	81	88

Step charts contain several invariances, for example interchanging all instances of left and right results in an equally plausible sequence of steps. To augment the amount of data available for training, we generate four instances of each chart, by mirroring left/right, up/down (or both). Doing so considerably improves performance in practice.

In addition to encoded audio, packs consist of metadata including a song's title, artist, a list of time-stamped tempo changes, and a time offset to align the recording to the tempos. They also contain information such as the chart difficulties and the name of the choreographer. Finally, the metadata contains a full list of steps, marking the measure and beat of each. To make this data easier to work with, we convert it to a canonical form consisting of (*beat*, *time*, *step*) tuples.

The charts in both datasets echo high-level rhythmic structure in the music. An increase in difficulty corresponds to increasing propensity for steps to appear at finer rhythmic subdivisions. Beginner charts tend to contain only quarter notes and eighth notes. Higher-difficulty charts reflect more complex rhythmic details in the music, featuring higher densities of eighth and sixteenth note steps (8th, 16th) as well as triplet patterns (12th, 24th) (Figure 6.4).

6.3 Problem Definition

A step can occur in up to 192 different locations (subdivisions) within each measure. However, measures contain roughly 6 steps on average. This level of sparsity makes it difficult to uncover patterns across long sequences of (mostly empty) frames via a single end-to-end

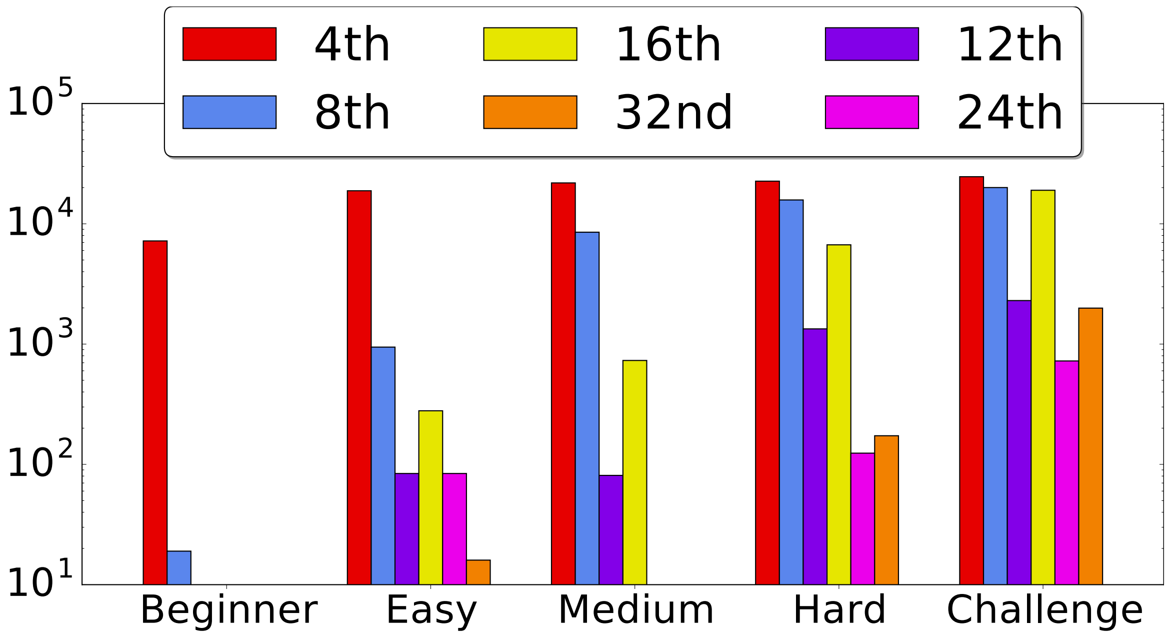


Figure 6.4: Number of steps per rhythmic subdivision by difficulty in the Fraxtil dataset.

sequential model. So, to make automatic DDR choreography tractable, we decompose it into two subtasks: step placement and step selection.

In *step placement*, our goal is to decide at what precise times to place steps. A step placement algorithm ingests raw audio features and outputs timestamps corresponding to steps. In addition to the audio signal, we provide step placement algorithms with a one-hot representation of the intended difficulty rating for the chart.

Step selection involves taking a discretized list of step times computed during step placement and mapping each of these to a DDR step. Our approach to this problem involves modeling the probability distribution $P(m_n|m_1,\ldots,m_{n-1})$ where m_n is the n^{th} step in the sequence. Some steps require that the player hit two or more arrows at once, a *jump*; or hold on one arrow for some duration, a *freeze* (Figure 6.2).

6.4 Methods

We now describe our specific solutions to the step placement and selection problems. Our basic pipeline works as follows: (1) extract an audio feature representation; (2) feed this representation into a step placement algorithm, which estimates probabilities that a ground truth step lies within that frame; (3) use a peak-picking process on this sequence of probabilities to identify the precise timestamps at which to place steps; and finally (4) given a sequence of timestamps, use a step selection algorithm to choose which steps to place at each time.

6.4.1 Audio Representation

Music files arrive as lossy encodings at $44.1kHz$. We decode the audio files into stereo PCM audio and average the two channels to produce a monophonic representation. We then compute a multiple-timescale short-time Fourier transform (STFT) using window lengths of 23*ms*, 46*ms*, and 93*ms* and a stride of 10*ms*. Shorter window sizes preserve low-level features such as pitch and timbre while larger window sizes provide more context for high-level features such as melody and rhythm [Hamel et al., 2012].

Using the ESSENTIA library [Bogdanov et al., 2013], we reduce the dimensionality of the STFT magnitude spectra to 80 frequency bands by applying a Mel-scale [Stevens et al., 1937] filterbank. We scale the filter outputs logarithmically to better represent human perception of loudness. Finally, we prepend and append seven frames of past and future context respectively to each frame.

For fixed-width methods, the final audio representation is a $15 \times 80 \times 3$ tensor. These correspond to the temporal width of 15 representing 150*ms* of audio context, 80 frequency bands, and 3 different window lengths. To better condition the data for learning, we normalize each frequency band to zero mean and unit variance. Our approach to acoustic feature representation closely follows the work of Schlüter and Böck [2014], who develop similar representations to

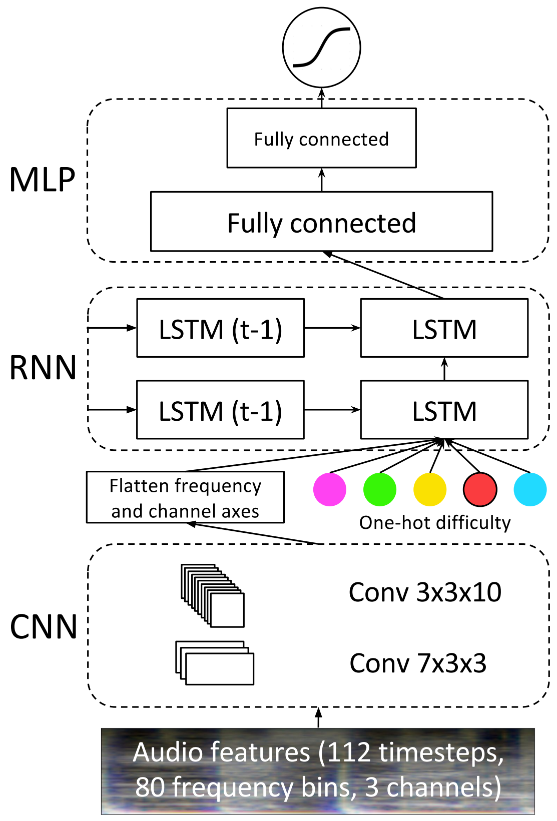


Figure 6.5: C-LSTM model used for step placement

perform onset detection with CNNs.

6.4.2 Step Placement

We consider several models to address the step placement task. Each model's output consists of a single sigmoid unit which estimates the probability that a step is placed. For all models, we augment the audio features with a one-hot representation of difficulty.

Following state-of-the-art work on onset detection Schlüter and Böck [2014], we adopt a convolutional neural network (CNN) architecture. This model consists of two convolutional layers followed by two fully connected layers. Our first convolutional layer has 10 filter kernels that are 7-wide in time and 3-wide in frequency. The second layer has 20 filter kernels that are 3-wide in time and 3-wide in frequency. We apply 1D max-pooling after each convolutional layer, only in the frequency dimension, with a width and stride of 3. Both convolutional layers use rectified linear units (ReLU) [Glorot et al., 2011]. Following the convolutional layers, we add two fully connected layers with rectifier activation functions and 256 and 128 nodes, respectively.

To improve upon the CNN, we propose a C-LSTM model, combining a convolutional encoding with an RNN that integrates information across longer windows of time. To encode the raw audio at each time step, we first apply two convolutional layers (of the same shape as the CNN) across the full unrolling length. The output of the second convolutional layer is a 3D tensor, which we flatten along the channel and frequency axes (preserving the temporal dimension). The flattened features at each time step then become the inputs to a two-layer RNN.

The C-LSTM contains long short-term memory (LSTM) units [Hochreiter and Schmidhuber, 1997] with forget gates [Gers and Schmidhuber, 2000]. The LSTM consists of 2 layers with 200 nodes each. Following the LSTM layers, we apply two fully connected ReLU layers of dimension 256 and 128. This architecture is depicted in Figure 6.5. We train this model using 100 unrollings for backpropagation through time.

A chart's intended difficulty influences decisions both about how many steps to place and where to place them. For low-difficulty charts, the average number of steps per second is less than one. In contrast, the highest-difficulty charts exceed seven steps per second. We trained all models both with and without conditioning on difficulty, and found the inclusion of this feature to be informative. We concatenate difficulty features to the flattened output of the CNN before feeding the vector to the fully connected (or LSTM) layers (Figure 6.5).[3] We initialize weight matrices following the scheme of Glorot and Bengio [2010].

Training Methodology We minimize binary cross-entropy with mini-batch stochastic gradient descent. For all models we train with batches of size 256, scaling down gradients when their l_2 norm exceeds 5. We apply 50% dropout following each LSTM and fully connected layer. For

[3]For LogReg and MLP, we add difficulty to input layer.

LSTM layers, we apply dropout in the input to output but not temporal directions, following best practices from [Zaremba et al., 2014, Lipton et al., 2016, Dai and Le, 2015]. Although the problem exhibits pronounced class imbalance (97% negatives), we achieved better results training on imbalanced data than with re-balancing schemes. We exclude all examples before the first step in the chart or after the last step as charts typically do not span the entire duration of the song.

For recurrent neural networks, the target at each frame is the ground truth value corresponding to that frame. We calculate updates using backpropagation through time with 100 steps of unrolling, equal to one second of audio or two beats on a typical track (120 BPM). We train all networks with early-stopping determined by the area under the precision-recall curve on validation data. All models satisfy this criteria within 12 hours of training on a single machine with an NVIDIA Tesla K40m GPU.

6.4.3 Peak Picking

Following standard practice for onset detection, we convert sequences of step probabilities into a discrete set of chosen placements via a peak-picking process. First we run our step placement algorithm over an entire song to assign the probabilities of a step occurring within each 10*ms* frame.[4] We then convolve this sequence of predicted probabilities with a Hamming window, smoothing the predictions and suppressing double-peaks from occurring within a short distance. Finally, we apply a constant threshold to choose which peaks are high enough (Figure 6.6). Because the number of peaks varies according to chart difficulty, we choose a different threshold per difficulty level. We consider predicted placements to be true positives if they lie within a $\pm 20ms$ window of a ground truth.

[4]In DDR, scores depend on the accuracy of a player's step timing. The highest scores require that a step is performed within 22.5*ms* of its appointed time; this suggests that a reasonable algorithm should place steps with an even finer level of granularity.

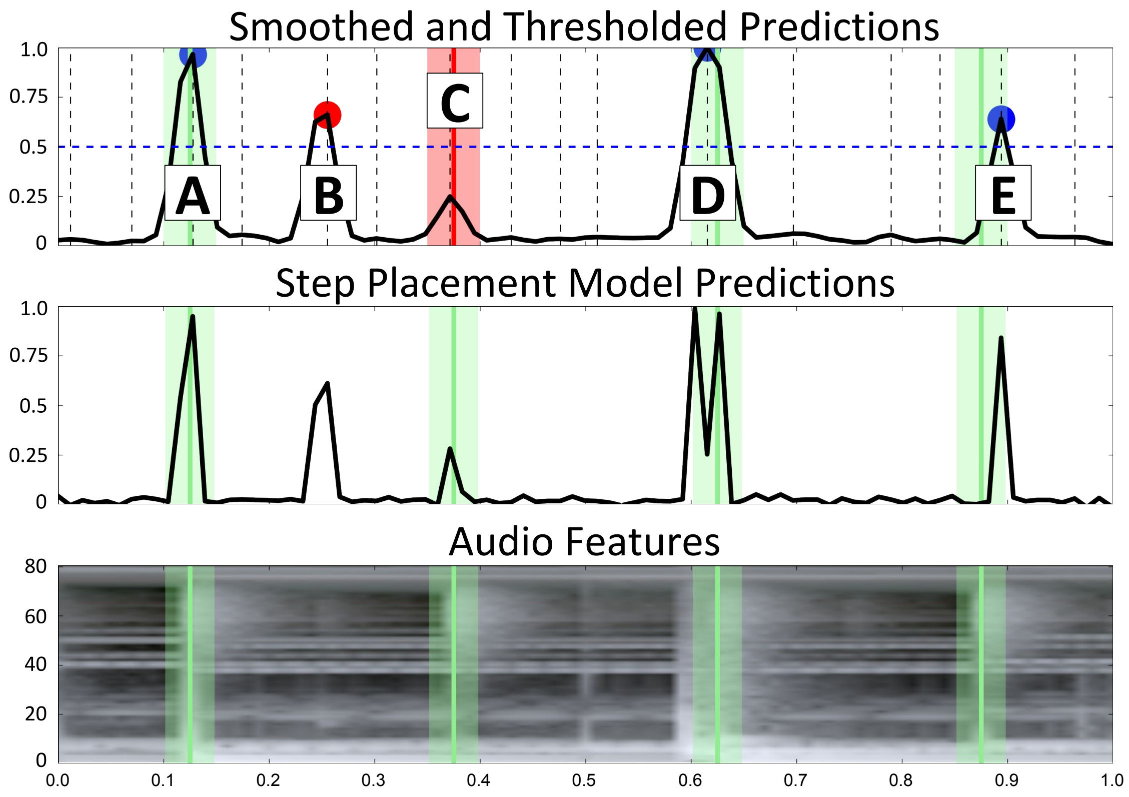


Figure 6.6: One second of peak picking. **Green:** Ground truth region **(A)**: true positive, **(B)**: false positive, **(C)**: false negative, **(D)**: two peaks smoothed to one by Hamming window, **(E)**: misaligned peak accepted as true positive by $\pm 20ms$ tolerance

6.4.4 Step Selection

We treat the step selection task as a sequence generation problem. Our approach follows related work in language modeling where RNNs are well-known to produce coherent text that captures long-range relationships [Mikolov et al., 2010, Sutskever et al., 2011, Sundermeyer et al., 2012].

Our LSTM model passes over the ground truth step placements and predicts the next token given the previous sequence of tokens. The output is a softmax distribution over the 256 possible steps. As input, we use a more compact *bag-of-arrows* representation containing 16 features (4 per arrow) to depict the previous step. For each arrow, the 4 corresponding features represent the states *on, off, hold,* and *release*. We found the bag-of-arrows to give equivalent performance to

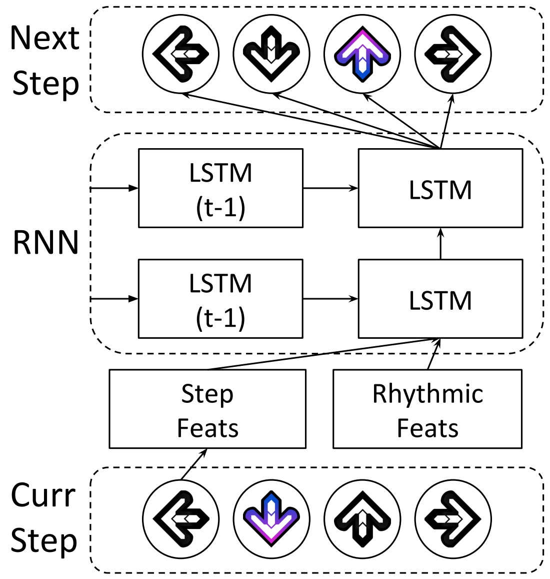


Figure 6.7: LSTM model used for step selection

the one-hot representation while requiring fewer parameters. We add an additional feature that functions as a *start* token to denote the first step of a chart. For this task, we use an LSTM with 2 layers of 128 cells each.

Finally, we provide additional musical context to the step selection models by conditioning on rhythmic features (Figure 6.7). To inform models of the non-uniform spacing of the step placements, we consider the following three features: (1) Δ-*time* adds two features representing the time since the previous step and the time until the next step; (2) Δ-*beat* adds two features representing the number of beats since the previous and until the next step; (3) *beat phase* adds four features representing which 16th note subdivision of the beat the current step most closely aligns to.

Training Methodology For all neural network models, we learn parameters by minimizing cross-entropy. We train with mini-batches of size 64, and scale gradients using the same scheme as for step placement. We use 50% dropout during training for both the MLP and RNN models in the same fashion as for step placement. We use 64 steps of unrolling, representing an average

Table 6.2: Results for step placement experiments

Model	**Dataset**	**PPL**	**AUC**	**F-score**^{c}	**F-score**^{m}
LogReg	Fraxtil	1.205	0.601	0.609	0.667
MLP	Fraxtil	1.097	0.659	0.665	0.726
CNN	Fraxtil	1.082	0.671	0.678	0.750
C-LSTM	Fraxtil	**1.070**	**0.682**	**0.681**	**0.756**
LogReg	ITG	1.123	0.599	0.634	0.652
MLP	ITG	1.090	0.637	0.671	0.704
CNN	ITG	1.083	0.677	0.689	0.719
C-LSTM	ITG	**1.072**	**0.680**	**0.697**	**0.721**

of 100 seconds for the easiest charts and 9 seconds for the hardest. We apply early-stopping determined by average per-step cross entropy on validation data. All models satisfy this criteria within 6 hours of training on a single machine with an NVIDIA Tesla K40m GPU.

6.5 Experiments

For both the Fraxtil and ITG datasets we apply 80%, 10%, 10% splits for training, validation, and test data, respectively. Because of correlation between charts for the same song of varying difficulty, we ensure that all charts for a particular song are grouped together in the same split.

6.5.1 Step Placement

We evaluate the performance of our step placement methods against baselines via the methodology outlined below.

Baselines To establish reasonable baselines for step placement, we first report the results of a logistic regressor (LogReg) trained on flattened audio features. We also report the performance of an MLP. Our MLP architecture contains two fully-connected layers of size 256 and 128, with

Table 6.3: Results for step selection experiments

Model	**Dataset**	**PPL**	**Accuracy**
KN5	Fraxtil	3.681	0.528
MLP5	Fraxtil	3.744	0.543
MLP5 + Δ-time	Fraxtil	3.495	0.553
MLP5 + Δ-beat + beat phase	Fraxtil	3.428	0.557
LSTM5	Fraxtil	3.583	0.558
LSTM5 + Δ-time	Fraxtil	3.188	0.584
LSTM5 + Δ-beat + beat phase	Fraxtil	3.185	0.581
LSTM64	Fraxtil	3.352	0.578
LSTM64 + Δ-time	Fraxtil	3.107	0.606
LSTM64 + Δ-beat + beat phase	Fraxtil	**3.011**	**0.613**
KN5	ITG	5.847	0.356
MLP5	ITG	5.312	0.376
MLP5 + Δ-time	ITG	4.792	0.402
MLP5 + Δ-beat + beat phase	ITG	4.786	0.401
LSTM5	ITG	5.040	0.407
LSTM5 + Δ-time	ITG	4.412	0.439
LSTM5 + Δ-beat + beat phase	ITG	4.447	0.441
LSTM64	ITG	4.780	0.426
LSTM64 + Δ-time	ITG	**4.284**	**0.454**
LSTM64 + Δ-beat + beat phase	ITG	4.342	0.444

rectifier nonlinearity applied to each layer. We apply dropout with probability 50% after each fully-connected layer during training. We model our CNN baseline on the method of Schlüter and Böck [2014], a state-of-the-art algorithm for onset detection.

Metrics We report each model's perplexity (PPL) averaged across each frame in each chart in the test data. Using the sparse step placements, we calculate the average per-chart area under the precision-recall curve (AUC). We average the best per-chart F-scores and report this value as F-score^c. We calculate the micro F-score across all charts and report this value as F-score^m.

In Table 6.2, we list the results of our experiments for step placement. For ITG, models were conditioned on not just difficulty but also a one-hot representation of chart author. For both datasets, the C-LSTM model performs the best by all evaluation metrics. Our models achieve significantly higher F-scores for harder difficulty step charts. On the Fraxtil dataset, the C-LSTM achieves an F-score^c of 0.844 for the hardest difficulty charts but only 0.389 for the lowest difficulty. The difficult charts contribute more to F-score^m calculations because they have more ground truth positives. We discuss these results further in Section 6.6.

6.5.2 Step Selection

Baselines For step selection, we compare the performance of the conditional LSTM to an *n*-gram model. Note that perplexity can be unbounded when a test set token is assigned probability 0 by the generative model. To protect the *n*-gram models against unbounded loss on previously unseen *n*-grams, we use modified Kneser-Ney smoothing [Chen and Goodman, 1998], following best practices in language modeling [Mikolov et al., 2010, Sutskever et al., 2011]. Specifically, we train a smoothed 5-gram model with backoff (*KN5*) as implemented in Stolcke [2002].

Following the work of Bengio et al. [2003] we also compare against a fixed-window 5-gram MLP which takes 4 bag-of-arrows-encoded steps as input and predicts the next step. The MLP contains two fully-connected layers with 256 and 128 nodes and 50% dropout after each layer during training. As with the LSTM, we train the MLP both with and without access to side features. In addition to the LSTM with 64 steps of unrolling, we train an LSTM with 5 steps of unrolling. These baselines show that the LSTM learns complex, long-range dependencies. They also demonstrate the discriminative information conferred by the Δ-time, Δ-beat, and beat phase features.

Metrics We report the average per-step perplexity, averaging scores calculated separately on each chart. We also report a per-token accuracy. We calculate accuracy by comparing the ground-truth step to the argmax over a model's predictive distribution given the previous sequence of ground-truth tokens. For a given chart, the per token accuracy is averaged across time steps. We produce final numbers by averaging scores across charts.

In Table 6.3 we present results for the step selection task. For the Fraxtil dataset, the best performing model was the LSTM conditioned on both Δ-beat and beat phase, while for ITG it was the LSTM conditioned on Δ-time. While conditioning on rhythm features was generally beneficial, the benefits of various features were not strictly additive. Representing Δ-beat and Δ-time as real numbers outperformed bucketed representations.

Additionally, we explored the possibility of incorporating more comprehensive representations of the audio into the step selection model. We considered a variety of representations, such as conditioning on CNN features learned from the step placement task. We also experimented with jointly learning a CNN audio encoder. In all cases, these approaches led to rapid overfitting and never approached the performance of the conditional LSTM generative model; perhaps a much larger dataset could support these approaches. Finally, we tried conditioning the step selection models on both difficulty and chart author but found these models to overfit quickly.

6.6 Discussion

Our experiments establish the feasibility of using machine learning to automatically generate high-quality DDR charts from raw audio. Our performance evaluations on both subtasks demonstrate the advantage of deep neural networks over classical approaches. For step placement, the best performing model is an LSTM with CNN encoder, an approach which has been used for speech recognition [Amodei et al., 2015], but, to our knowledge, never for music-related tasks. We noticed that by all metrics, our models perform better on higher-difficulty charts. Likely, this owes to the comparative class imbalance of the lower difficulty charts.

The superior performance of LSTMs over fixed-window approaches on step selection suggests both that DDR charts exhibit long range dependencies and that recurrent neural networks can exploit this complex structure. In addition to reporting quantitative results, we visualize the step selection model's next-step predictions. Here, we give the entire ground truth sequence as

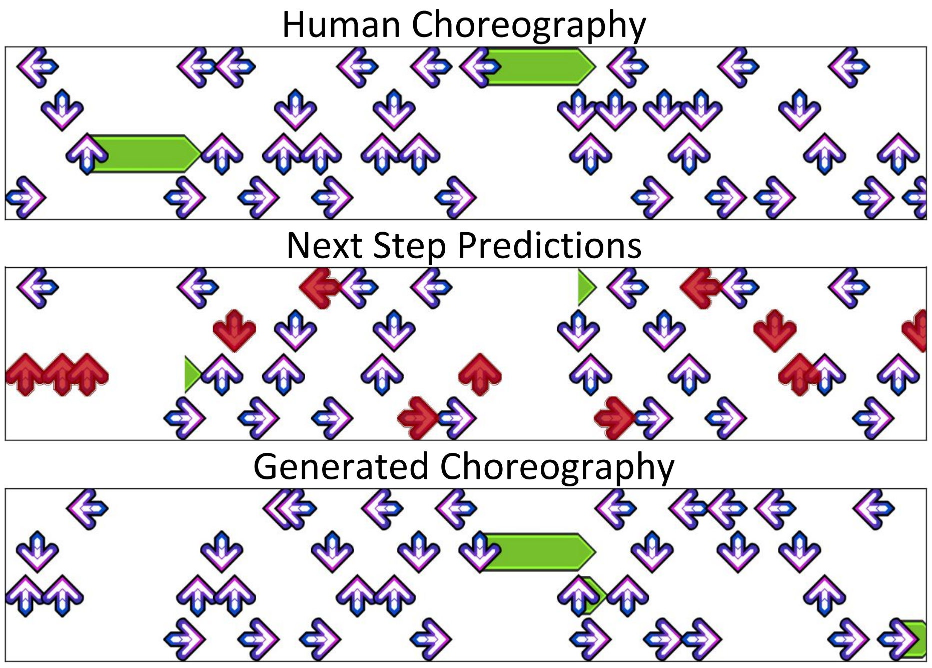


Figure 6.8: **Top:** A real step chart from the *Fraxtil* dataset on the song *Anamanaguchi - Mess*. **Middle:** One-step lookahead *predictions* for the LSTM model, given Fraxtil's choreography as input. The model predicts the next step with high accuracy (errors in red). **Bottom:** Choreography generated by conditional LSTM model.

input but show the predicted next step at each time. We also visualize a generated choreography, where each sampled output from the LSTM is fed in as the subsequent input (Figure 6.8). We note the high accuracy of the model's predictions and qualitative similarity of the generated sequence to Fraxtil's choreography.

For step selection, we notice that modeling the Fraxtil dataset choreography appears to be easy compared to the multi-author ITG dataset. We believe this owes to the distinctiveness of author styles. Because we have so many step charts for Fraxtil, the network is able to closely mimic his patterns. While the ITG dataset contains multiple charts per author, none are so prolific as Fraxtil.

We released a public demo[5] using our most promising models as measured by our quantitative evaluation. Players upload an audio file, select a difficulty rating and receive a step

[5]http://deepx.ucsd.edu/ddc

chart for use in the StepMania DDR simulator. Our demo produces a step chart for a 3 minute song in about 5 seconds using an NVIDIA Tesla K40c GPU. At time of writing, 220 players have produced 1370 step charts with the demo. We also solicited feedback, on a scale of 1-5, for player "satisfaction" with the demo results. The 22 respondents reported an average satisfaction of 3.87.

A promising direction for future work is to make the selection algorithm audio-aware. We know qualitatively that elements in the ground truth choreography tend to coincide with specific musical events: jumps are used to emphasize accents in a rhythm; freezes are used to transition from regions of high rhythmic intensity to more ambient sections.

DDR choreography might also benefit from an end-to-end approach, in which a model simultaneously places steps and selects them. The primary obstacle here is data sparsity at any sufficiently high feature rate. At $100Hz$, about 97% of labels are null. So in 100 time-steps of unrolling, an RNN might only encounter 3 ground truth steps.

We demonstrate that step selection methods are improved by incorporating Δ-beat and beat phase features, however our current pipeline does not produce this information. In lieu of manual tempo input, we are restricted to using Δ-time features when executing our pipeline on unseen recordings. If we trained a model to detect beat phase, we would be able to use these features for step selection.

6.7 Related Work

Several academic papers address DDR. These include anthropological studies [Hoysniemi, 2006, Behrenshausen, 2007] and two papers that describe approaches to automated choreography. The first, called *Dancing Monkeys*, uses rule-based methods for both step placement and step selection [O'Keeffe, 2003]. The second employs genetic algorithms for step selection, optimizing an ad-hoc fitness function [Nogaj, 2005]. Neither establishes reproducible evaluation methodology or learns the semantics of steps from data.

Our step placement task closely resembles the classic problem of musical onset detection [Bello et al., 2005, Dixon, 2006]. Several onset detection papers investigate modern deep learning methodology. Eyben et al. [2010] employ bidirectional LSTMs (BLSTMs) for onset detection; Marchi et al. [2014] improve upon this work, developing a rich multi-resolution feature representation; Schlüter and Böck [2014] demonstrate a CNN-based approach (against which we compare) that performs competitively with the prior BLSTM work. Neural networks are widely used on a range of other MIR tasks, including musical chord detection [Humphrey and Bello, 2012, Boulanger-Lewandowski et al., 2013a] and boundary detection [Ullrich et al., 2014], another transient audio phenomenon.

Our step selection problem resembles the classic natural language processing task of statistical language modeling. Classical methods, which we consider, include *n*-gram distributions [Chen and Goodman, 1998, Rosenfeld, 2000]. Bengio et al. [2003] demonstrate an approach to language modeling using neural networks with fixed-length context. More recently, RNNs have demonstrated superior performance to fixed-window approaches [Mikolov et al., 2010, Sundermeyer et al., 2012, Sutskever et al., 2011]. LSTMs are also capable of modeling language at the character level [Karpathy et al., 2015, Kim et al., 2016]. While a thorough explanation of modern RNNs exceeds the scope of this paper, we point to two comprehensive reviews of the literature [Lipton et al., 2015, Greff et al., 2016]. Several papers investigate neural networks for single-note melody generation [Bharucha and Todd, 1989, Eck, 2002, Chu et al., 2016, Hadjeres and Pachet, 2017] and polyphonic melody generation [Boulanger-Lewandowski et al., 2012].

Learning to choreograph requires predicting both the *timing* and the *type* of events in relation to a piece of music. In that respect, our task is similar to audio sequence transduction tasks, such as musical transcription and speech recognition. RNNs currently yield state-of-the-art performance for musical transcription [Böck and Schedl, 2012, Boulanger-Lewandowski et al., 2013b, Sigtia et al., 2016]. RNNs are widely used for speech recognition [Graves and Jaitly, 2014, Graves et al., 2006, 2013, Sainath et al., 2015], and the state-of-the-art method [Amodei et al., 2015] combines convolutional and recurrent networks. While our work is methodologically similar, it differs from the above in that we consider an entirely different application.

6.8 Conclusions

By combining insights from musical onset detection and statistical language modeling, we have designed and evaluated a number of deep learning methods for *learning to choreograph*. We have introduced standardized datasets and reproducible evaluation methodology in the hope of encouraging wider investigation into this and related problems. We emphasize that the sheer volume of available step charts presents a rare opportunity for MIR: access to large amounts of high-quality annotated data. This data could help to spur innovation for several MIR tasks, including onset detection, beat tracking, and tempo detection.

6.9 Acknowledgements

I would like to thank Jeff Donahue, Shlomo Dubnov, Jennifer Hsu, Mohsen Malmir, Miller Puckette, Adith Swaminathan and Sharad Vikram for their helpful feedback on this work. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562. GPUs used for this research were graciously donated by the NVIDIA Corporation.

Chapter 6, in full, is a reprint of the material as it appears in *Dance Dance Convolution*. 2017. Lipton, Zachary C.; McAuley, Julian [Donahue et al., 2017]. International Conference on Machine Learning 2017. The dissertation/thesis author was the primary investigator and author of this paper.

Chapter 7

Other interactive systems

In this chapter, I discuss other interactive systems I have built which allow humans to pilot and curate neural network generative models. These systems merge aspects of traditional computer music interfaces (e.g. sequencers) with powerful generative models. In Section 7.1, I discuss a demo which uses my WaveGAN generative model (Chapter 4) to produce a procedural drum machine. In Section 7.2, I discuss *Neural Loops*, a factorized approach to modeling $P(\text{music})$ which combines three different neural network generative models to procedurally create four-bar musical loops. In Section 7.3, I discuss an interface I built for my research on decomposing the *latent spaces* of neural network generative models in semantically-meaningful ways.

7.1 DrumGAN

I built a demo interface allowing musicians to explore our WaveGAN audio generative model (Chapter 4) trained on drum one-shot clips (DrumGAN) (link: https://chrisdonahue.com/wavegan). It pairs a traditional 16-step drum sequencer with our DrumGAN synthesis method, allowing musicians to independently manipulate both the symbolic and acoustic parts of a drum loop. It also allows the user to download the sounds generated by WaveGAN for use in another

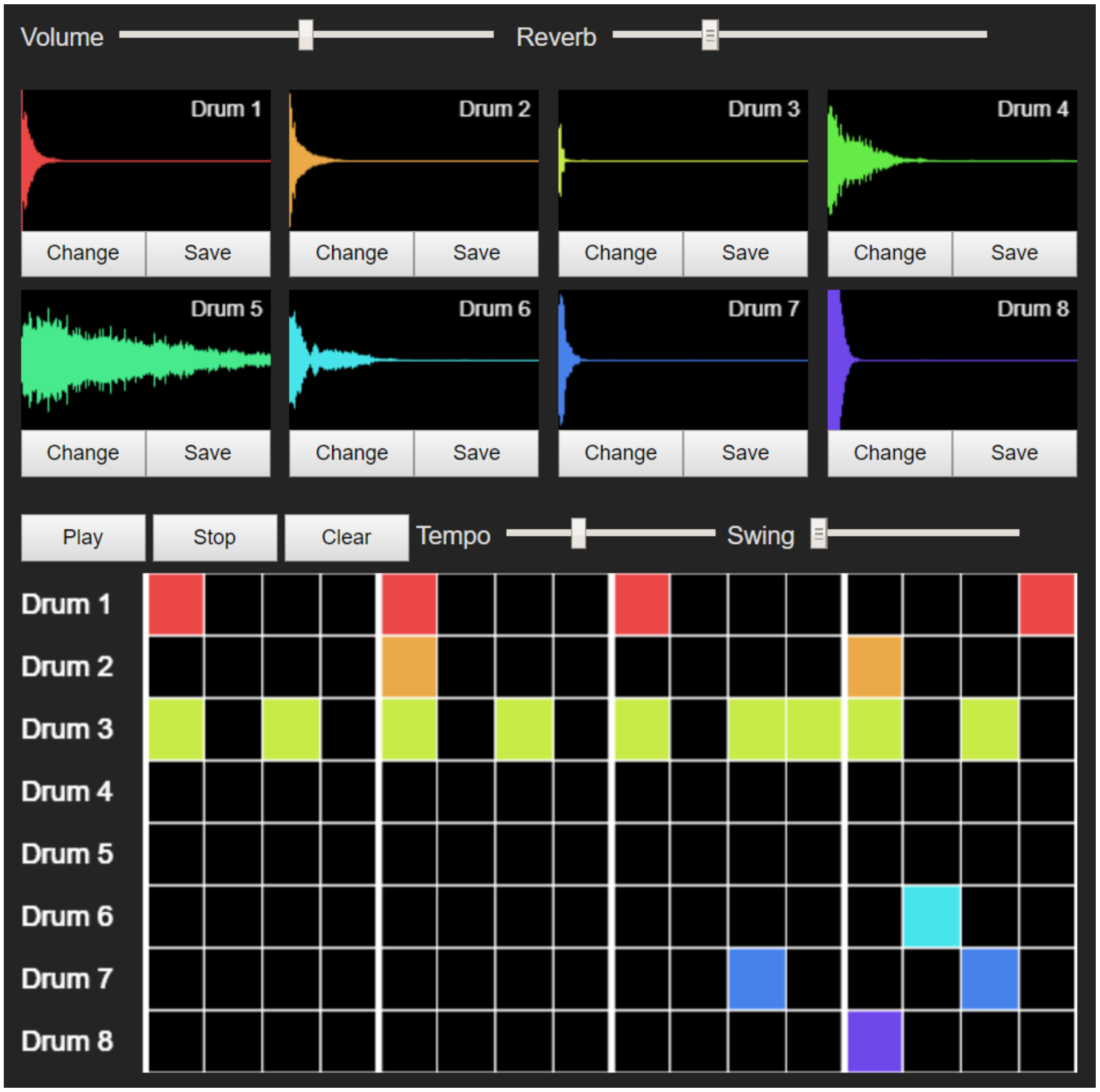


Figure 7.1: Interface for a demo of DrumGAN, which combines a WaveGAN generative model of drum one-shots with a traditional step sequencer.

digital audio workstation. This tool requires some amount of musical expertise as it tasks users with manually creating symbolic drum patterns. In Section 7.2 we describe a similar demo which incorporates a model capable of automatically generating these patterns.

7.2 Neural Loops

Neural Loops (Figure 7.2, video demonstration https://youtu.be/7A_jur5ncHI) is an interactive system for creating four-bar musical loops using machine learning. It was created in collaboration with Vibert Thio. Neural Loops uses a factorized approach to modeling $P(\text{music})$ (Section 2.2). Specifically, it combines the MusicVAE method [Roberts et al., 2018] to generate four-bar trio scores with my work on generating drum one-shots (WaveGAN) [Donahue et al., 2019a] and tonal instrument timbres (GANSynth) [Engel et al., 2019].

Because we focus on a subset of $P(\text{music})$ which encapsulates four-bar trios performed by an ensemble consisting of a melody, bass line, and drum kit, I will instead refer to the distribution as $P(\text{loop})$. The factorization we explore here is

$$P(\text{loop}) = P(\text{score}) \cdot P(\text{melody timbre}) \cdot P(\text{bass timbre}) \cdot P(\text{drumkit timbre}). \tag{7.1}$$

Intuitively, this means that we have independent control over the trio score, the tonal instrument (melody/bass) timbres, and the percussive timbres (Figure 7.3). One assumption we are making in Equation (7.1) are that the instrument timbres are statistically independent of the score and vice versa, e.g. $P(\text{melody timbre} \mid \text{score}) = P(\text{melody timbre})$.

In Figure 7.2, I show the interface we built for Neural Loops. The interface consists of a combination of traditional musical interface controls like volume and tempo faders, and "smart" buttons which resample from the factorized distribution of Equation (7.1). Using the smart buttons, the user can independently modify the individual instrument timbres (melody, bass, or any of 9 drum kit sounds) and individual instrument scores.

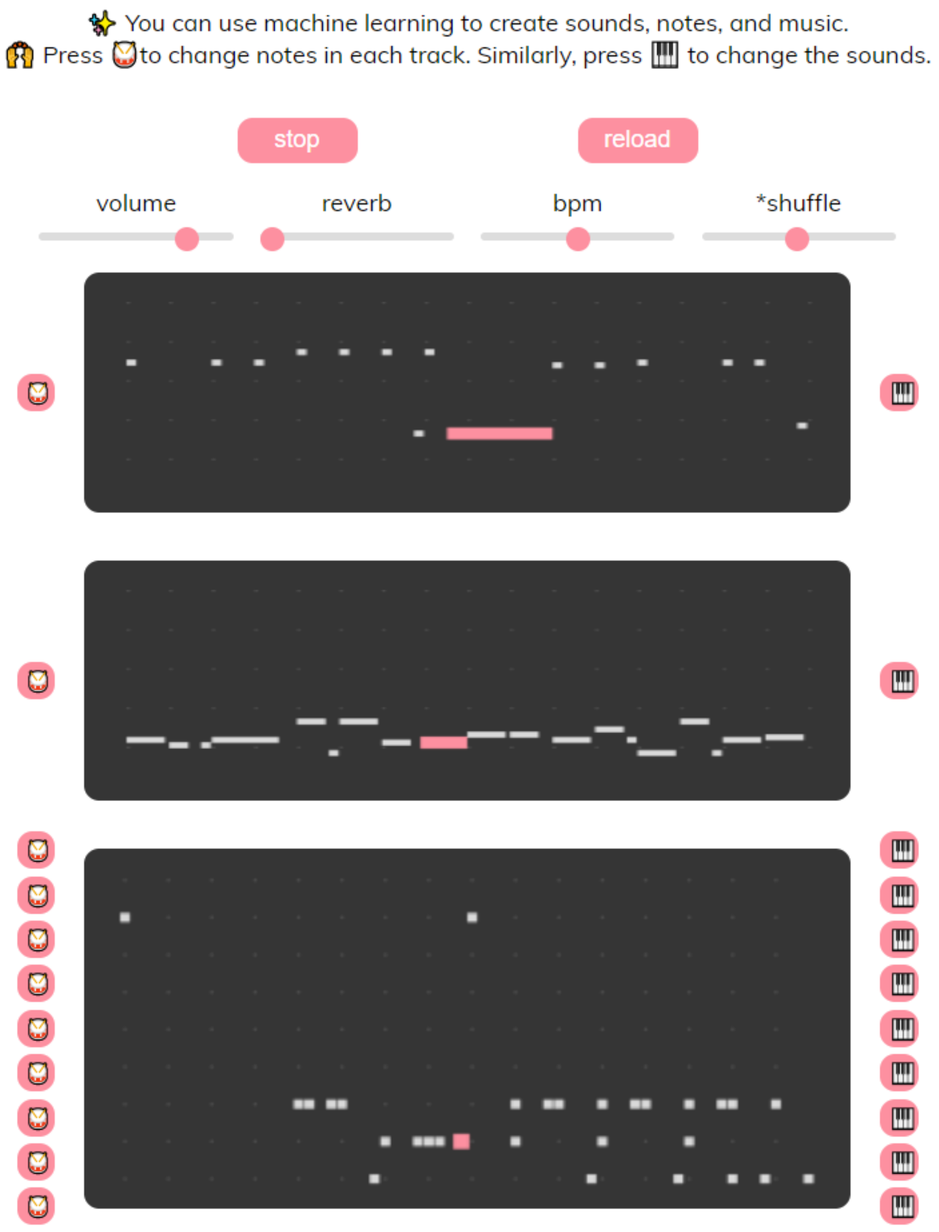

✨ You can use machine learning to create sounds, notes, and music.
🙌 Press 🥁 to change notes in each track. Similarly, press 🎹 to change the sounds.
stop
reload
volume
reverb
bpm
*shuffle

Figure 7.2: The interface for *Neural Loops*, which combines multiple neural network generative models with a traditional musical sequencer to produce an interactive tool for exploring four-bar musical trios.

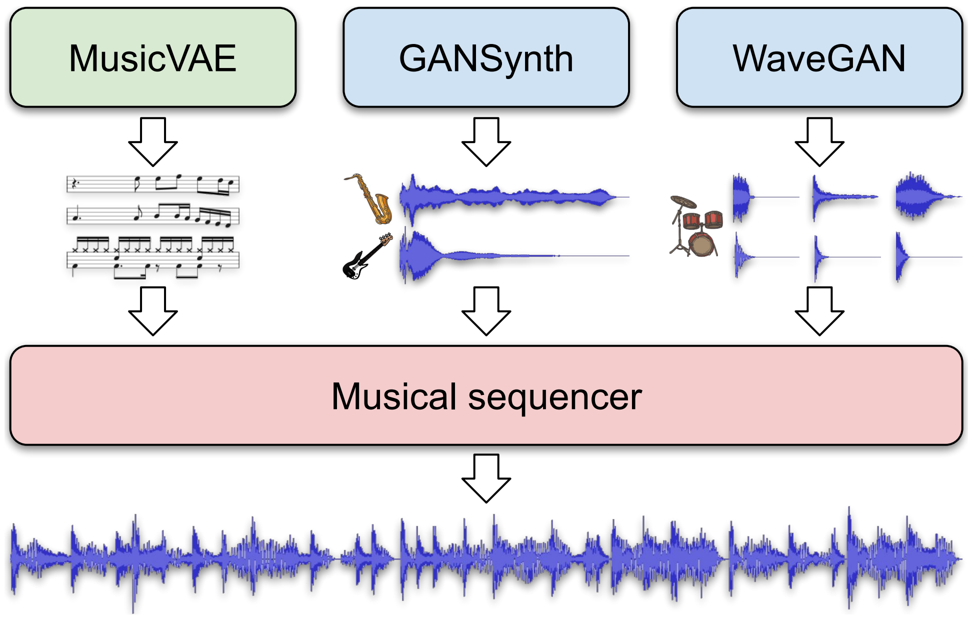


Figure 7.3: Schematic diagram for *Neural Loops* illustrating the generative models used as part of a factorized approach at modeling four-bar musical waveforms.

While Neural Loops is—for now—more a proof of concept than fully-fledged tool, I see it as a first step towards a predictive digital audio workstation (DAW). Such a DAW could allow human pilots to explore a continuum between full automation (creating a complete track at the press of a button), and no automation (reminiscent of current DAW workflows). The point along this continuum which a user finds to be the most productive would be a function of both their musical experience level and intentions. Such tools would allow non-musicians to create music and also modify the workflow of expert musicians to enhance their productivity.

7.3 Semantically-decomposed GANs

I built an interface to accompany my work on semantically decomposing the latent spaces of generative adversarial networks [Donahue et al., 2018b]. The goal of this work was to factorize the distribution of multimedia into the parts corresponding to some semantically-salient attribute, and the parts corresponding to everything *except* that attribute, which we call *contingent* aspects.

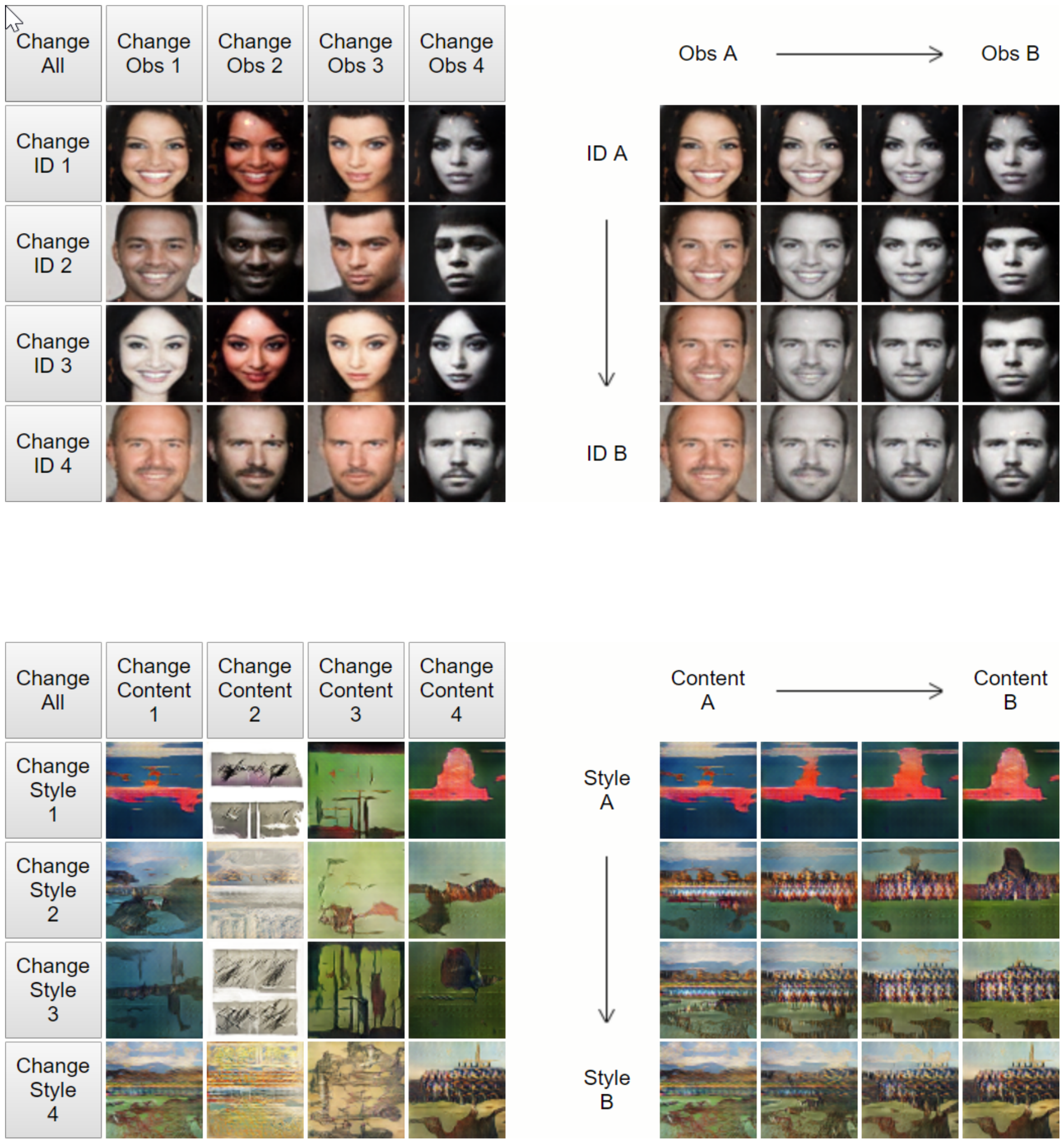


Figure 7.4: The factorized latent spaces of semantically-decomposed generative adversarial networks allow for independent manipulation of salient and contingent aspects of multimedia such as face portraits and visual artworks.

For example, we factorized the distribution of face portraits into a portion pertaining to the *identity* of the subject depicted, and a portion pertaining to all of the contingent factors of a face portrait that can vary besides identity (e.g. lighting, angle). We also attempted to factorize the distribution of visual art into portions corresponding to *style* and the contingent, where we loosely represent style as the identity of an individual artist.

The methodological details can be found in [Donahue et al., 2018b], but here I want to discuss the interactive interfaces we built for these two case studies (Figure 7.4). Both allow the user to independently manipulate the salient attribute (identity for faces and style for visual art) and the other contingent aspects (left portion of each interface). They also allow the user to *interpolate* between these attributes (right portion of each interface). While these examples focus on non-musical types of multimedia, they further help to exemplify a common thread of all of my research in generative modeling. Namely, that factorized probabilistic models allow for convenient interfaces that allow humans to intervene and benefit from the generation procedure of the model. This is almost always more useful than interfaces which generate infinite amounts of content at the push of a single button.

Part IV

Conclusion

Chapter 8

Conclusion

In this dissertation I have described my research which enables new types of musical interactions by using predictive and generative machine learning models. My work has advanced the state-of-the-art in unsupervised generative modeling of both multi-instrumental symbolic music and audio waveforms. Additionally, I have built numerous interfaces which allow users to benefit from what these predictive models have learned about human-composed music. Perhaps most illustrative of my goal of lowering the barrier to entry for music creation is my work on *Piano Genie* (Chapter 5, demo: https://piano-genie.glitch.me), which grants non-musicians a degree of piano improvisation literacy.

As I continue my research into enabling these new types of musical interactions, I emphasize both the practical and philosophical importance of building systems which are primarily human-driven. Researchers in music generation have conceived many different quantitative evaluation metrics by which to compare our systems, but ultimately the only true metric is the degree of engagement and enjoyment our systems bring to humans. Hence, it is important that we continue to design systems with human users in mind, allowing artificial intelligence to become yet another tool in the toolbox for human musicians—much like electronics before it.

Bibliography

Moray Allan and Christopher Williams. Harmonising chorales by probabilistic inference. In *NIPS*, 2005.

Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al. Deep Speech 2: End-to-end speech recognition in english and mandarin. In *ICML*, 2015.

Mathieu Andreux and Stephane Mallat. Music generation and transformation with moment matching-scattering inverse networks. In *ISMIR*, 2018.

Sercan Arik, Mike Chrzanowski, Adam Coates, Gregory Diamos, Andrew Gibiansky, Yongguo Kang, Xian Li, John Miller, Jonathan Raiman, Shubho Sengupta, et al. Deep Voice: Real-time neural text-to-speech. *arXiv:1702.07825*, 2017.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. In *ICML*, 2017.

Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. In *ICLR*, 2017.

Bryan G Behrenshausen. Toward a (kin) aesthetic of video gaming the case of Dance Dance Revolution. *Games and Culture*, 2007.

Juan Pablo Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B Sandler. A tutorial on onset detection in music signals. *IEEE Transactions on Speech and Audio Processing*, 2005.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *JMLR*, 2003.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432*, 2013.

David Berthelot, Tom Schumm, and Luke Metz. BEGAN: Boundary equilibrium generative adversarial networks. *arXiv:1703.10717*, 2017.

Frédéric Bevilacqua, Rémy Müller, and Norbert Schnell. MnM: a Max/MSP mapping toolbox.

In *NIME*, 2005.

Jamshed J Bharucha and Peter M Todd. Modeling the perception of tonal structure with neural nets. *Computer Music Journal*, 1989.

Sebastian Böck and Markus Schedl. Polyphonic piano note transcription with recurrent neural networks. In *ICASSP*, 2012.

Peter Boesman. Bird recordings. https://www.xeno-canto.org/contributor/OOECIWCSWV, 2018. Accessed: 2018-01-08.

Dmitry Bogdanov, Nicolas Wack, Emilia Gómez, Sankalp Gulati, Perfecto Herrera, Oscar Mayor, Gerard Roma, Justin Salamon, José R Zapata, and Xavier Serra. Essentia: An audio analysis library for music information retrieval. In *ISMIR*, 2013.

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *ICML*, 2012.

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Audio chord recognition with recurrent neural networks. In *ISMIR*, 2013a.

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. High-dimensional sequence transduction. In *ICASSP*, 2013b.

Johannes Buchner. Synthetic speech commands dataset. https://www.kaggle.com/jbuchner/synthetic-speech-commands-dataset, 2017. Accessed: 2017-01-15.

Lele Chen, Sudhanshu Srivastava, Zhiyao Duan, and Chenliang Xu. Deep cross-modal audio-visual generation. In *ACM Multimedia Thematic Workshops*, 2017.

Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard University, 1998.

Hang Chu, Raquel Urtasun, and Sanja Fidler. Song from PI: A musically plausible network for pop music generation. *arXiv:1611.03477*, 2016.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS Workshops*, 2014.

Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *NIPS*, 2015.

Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. *arXiv:1901.02860*, 2019.

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

Sander Dieleman, Aäron van den Oord, and Karen Simonyan. The challenge of realistic music generation: modelling raw audio at scale. In *NeurIPS*, 2018.

Simon Dixon. Onset detection revisited. In *DAFx*, 2006.

Chris Donahue, Zachary C Lipton, and Julian McAuley. Dance Dance Convolution. In *ICML*, 2017.

Chris Donahue, Bo Li, and Rohit Prabhavalkar. Exploring speech enhancement with generative adversarial networks for robust speech recognition. In *ICASSP*, 2018a.

Chris Donahue, Zachary C Lipton, Akshay Balsubramani, and Julian McAuley. Semantically decomposing the latent spaces of generative adversarial networks. In *ICLR*, 2018b.

Chris Donahue, Huanru Henry Mao, and Julian McAuley. The NES Music Database: A multi-instrumental dataset with expressive performance attributes. In *ISMIR*, 2018c.

Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. In *ICLR*, 2019a.

Chris Donahue, Ian Simon, and Sander Dieleman. Piano Genie. In *ACM IUI*, 2019b.

Hao-Wen Dong and Yi-Hsuan Yang. Convolutional generative adversarial networks with binary neurons for polyphonic music generation. In *ISMIR*, 2018.

W Jay Dowling. Scale and contour: Two components of a theory of memory for melodies. *Psychological review*, 1978.

Homer Dudley. Remaking speech. *The Journal of the Acoustical Society of America*, 1939.

Douglas Eck. A first look at music composition using LSTM recurrent neural networks. Technical report, IDSIA, 2002.

Douglas Eck and Jürgen Schmidhuber. Finding temporal structure in music: Blues improvisation with LSTM recurrent networks. In *Neural Networks for Signal Processing*, 2002.

Zohar Eitan, Asi Schupak, Alex Gotler, and Lawrence E Marks. Lower pitch is larger, yet falling pitches shrink. *Experimental psychology*, 2014.

Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with WaveNet autoencoders. In *ICML*, 2017.

Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. GANSynth: Adversarial neural audio synthesis. In *ICLR*, 2019.

Florian Eyben, Sebastian Böck, Björn W Schuller, and Alex Graves. Universal onset detection with bidirectional long short-term memory neural networks. In *ISMIR*, 2010.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *ACL*, 2018.

Zhe-Cheng Fan, Yen-Lin Lai, and Jyh-Shing Roger Jang. SVSGAN: Singing voice separation via generative adversarial network. *arXiv:1710.11428*, 2017.

Rebecca Fiebrink, Dan Trueman, and Perry R Cook. A meta-instrument for interactive, on-the-fly machine learning. In *NIME*, 2009.

John S Garofolo, Lori F Lamel, William M Fisher, Jonathan G Fiscus, David S Pallett, Nancy L Dahlgren, and Victor Zue. TIMIT acoustic-phonetic continuous speech corpus. *Linguistic data consortium*, 1993.

Felix Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *International Joint Conference on Neural Networks*, 2000.

Andrew Gibiansky, Sercan Arik, Gregory Diamos, John Miller, Kainan Peng, Wei Ping, Jonathan Raiman, and Yanqi Zhou. Deep Voice 2: Multi-speaker neural text-to-speech. In *NIPS*, 2017.

Nicholas Gillian and Benjamin Knapp. A machine learning toolbox for musician computer interaction. In *NIME*, 2011.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *AISTATS*, 2011.

Rolf Inge Godøy and Alexander Refsum Jensenius. Body movement in music information retrieval. In *ISMIR*, 2009.

Rolf Inge Godøy, Egil Haga, and Alexander Refsum Jensenius. Playing "air instruments": mimicry of sound-producing gestures by novices and experts. In *International Gesture Workshop*, 2005.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. In *NIPS*, 2014.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *ICML*, 2014.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *ICML*, 2006.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, 2013.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

Daniel W Griffin and Jae S Lim. Signal estimation from modified short-time Fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1984.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of Wasserstein GANs. In *NIPS*, 2017.

Gaëtan Hadjeres and François Pachet. DeepBach: A steerable model for Bach chorales generation. In *ICML*, 2017.

Philippe Hamel, Yoshua Bengio, and Douglas Eck. Building musically-relevant audio features through multiple timescale representations. In *ISMIR*, 2012.

Harmonix. *The Axe: Titans of Classic Rock*. Game, 1998.

Harmonix. *Guitar Hero*. Game, 2005.

Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *ICLR*, 2019.

Shawn Hershey, Sourish Chaudhuri, Daniel PW Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Rif A Saurous, Bryan Seybold, et al. CNN architectures for large-scale audio classification. In *ICASSP*, 2017.

Hermann Hild, Johannes Feulner, and Wolfram Menzel. HARMONET: A neural net for harmonizing chorales in the style of JS Bach. In *NIPS*, 1992.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

Johanna Hoysniemi. International survey on the Dance Dance Revolution game. *Computers in Entertainment*, 2006.

Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Douglas Eck. Counterpoint by convolution. In *ISMIR*, 2017.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music Transformer: Generating music with long-term structure. In *ICLR*, 2019.

Eric J Humphrey and Juan Pablo Bello. Rethinking automatic chord recognition with convolutional neural networks. In *International Conference on Machine Learning and Applications*, 2012.

Andrew J Hunt and Alan W Black. Unit selection in a concatenative speech synthesis system using a large speech database. In *ICASSP*, 1996.

David Huron. The melodic arch in western folksongs. *Computing in Musicology*, 1996.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

Keith Ito. The LJ Speech dataset. https://keithito.com/LJ-Speech-Dataset/, 2017.

Daniel D Johnson. Generating polyphonic music using tied parallel networks. In *International Conference on Evolutionary and Biologically Inspired Music and Art*, 2017.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv:1506.02078*, 2015.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *ICLR*, 2018.

Tejaswinee Kelkar and Alexander Refsum Jensenius. Representation strategies in two-handed melodic sound-tracing. In *International Conference on Movement Computing*, 2017.

Tejaswinee Kelkar, Udit Roy, and Alexander Refsum Jensenius. Evaluating a collection of sound-tracing data of melodic phrases. In *ISMIR*, 2018.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Tetsuro Kitahara, Sergio I Giraldo, and Rafael Ramírez. JamSketch: A drawing-based real-time evolutionary improvisation support system. In *NIME*, 2017.

Olivier Lartilot. *Sound Tracer*, 2018.

Jonathan Le Roux, Hirokazu Kameoka, Nobutaka Ono, and Shigeki Sagayama. Fast signal reconstruction from magnitude STFT spectrogram based on spectrogram consistency. In *DAFx*, 2010.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

Jongpil Lee, Jiyoung Park, Keunhyoung Luke Kim, and Juhan Nam. Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms. In *SMC*, 2017.

Michael Lee, Adrian Freed, and David Wessel. Neural networks for simultaneous classification and parameter estimation in musical instrument control. In *Adaptive and Learning Systems*, 1992.

Rensis Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 1932.

Zhen-Hua Ling, Shi-Yin Kang, Heiga Zen, Andrew Senior, Mike Schuster, Xiao-Jun Qian, Helen M Meng, and Li Deng. Deep learning for acoustic modeling in parametric speech generation: A systematic review of existing techniques and future trends. *IEEE Signal Processing Magazine*, 2015.

Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv:1506.00019*, 2015.

Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzell. Learning to diagnose with LSTM recurrent neural networks. In *ICLR*, 2016.

Huanru Henry Mao, Taylor Shin, and Garrison W Cottrell. DeepJ: Style-specific music generation. In *International Conference on Semantic Computing*, 2018.

Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *ICCV*, 2017.

Erik Marchi, Giacomo Ferroni, Florian Eyben, Leonardo Gabrielli, Stefano Squartini, and Bjorn Schuller. Multi-resolution linear prediction based features for audio onset detection with bidirectional LSTM neural networks. In *ICASSP*, 2014.

Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv:1511.05440*, 2015.

Brian McFee, Jong Wook Kim, Mark Cartwright, Justin Salamon, Rachel M Bittner, and Juan Pablo Bello. Open-source practices for music signal processing research: Recommendations for transparent, sustainable, and reproducible audio research. *IEEE Signal Processing Magazine*, 2019.

Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. SampleRNN: An unconditional end-to-end neural audio generation model. In *ICLR*, 2017.

Daniel Michelsanti and Zheng-Hua Tan. Conditional generative adversarial networks for speech enhancement and noise-robust speaker verification. In *INTERSPEECH*, 2017.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, 2010.

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

Masanori Morise, Fumiya Yokomori, and Kenji Ozawa. WORLD: a vocoder-based high-quality speech synthesis system for real-time applications. *IEICE Transactions on Information and Systems*, 2016.

Eric Moulines and Francis Charpentier. Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones. *Speech communication*, 1990.

Michael C Mozer. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 1994.

Paarth Neekhara, Chris Donahue, Miller Puckette, Shlomo Dubnov, and Julian McAuley. Expediting TTS synthesis with adversarial vocoding. *arXiv:1904.07944*, 2019.

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

Adam Nogaj. A genetic algorithm for determining optimal step patterns in Dance Dance Revolution. Technical report, State University of New York at Fredonia, 2005.

Kristian Nymoen, Baptiste Caramiaux, Mariusz Kozak, and Jim Torresen. Analyzing sound tracings: a multimodal approach to music information retrieval. In *ACM Workshop on Music Information Retrieval with User-centered and Multimodal Strategies*, 2011.

Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

Karl O'Keeffe. Dancing monkeys (automated creation of step files for Dance Dance Revolution). Technical report, Imperial College London, 2003.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 2010.

Denys Parsons. *The directory of tunes and musical themes*. S. Brown, 1975.

Santiago Pascual, Antonio Bonafonte, and Joan Serrà. SEGAN: Speech enhancement generative adversarial network. In *INTERSPEECH*, 2017.

Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016.

Wei Ping, Kainan Peng, Andrew Gibiansky, Sercan Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John Miller. Deep Voice 3: Scaling text-to-speech with convolutional sequence learning. In *ICLR*, 2018.

Ryan Prenger, Rafael Valle, and Bryan Catanzaro. WaveGlow: A flow-based generative network for speech synthesis. In *ICASSP*, 2018.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

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

Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. In *ICML*, 2018.

Ronald Rosenfeld. Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, 2000.

Udit Roy, Tejaswinee Kelkar, and Bipin Indurkhya. TrAP: An interactive system to generate valid raga phrases from sound-tracings. In *NIME*, 2014.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, UC San Diego Institute for Cognitive Science, 1985.

Tara N Sainath, Ron J Weiss, Andrew Senior, Kevin W Wilson, and Oriol Vinyals. Learning the speech front-end with raw waveform CLDNNs. In *INTERSPEECH*, 2015.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *NIPS*, 2016.

Jan Schlüter and Sebastian Böck. Improved musical onset detection with convolutional neural networks. In *ICASSP*, 2014.

Hugo Scurto and Rebecca Fiebrink. Grab-and-play mapping: Creative machine learning approaches for musical inclusion and exploration. In *ICMC*, 2016.

Jonathan Shen, Ruoming Pang, Ron J Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, RJ Skerry-Ryan, et al. Natural TTS synthesis by conditioning WaveNet on mel spectrogram predictions. In *ICASSP*, 2018.

Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *CVPR*, 2017.

Siddharth Sigtia, Emmanouil Benetos, and Simon Dixon. An end-to-end neural network for polyphonic piano music transcription. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2016.

Ian Simon and Sageev Oore. Performance RNN: Generating music with expressive timing and dynamics. https://magenta.tensorflow.org/performance-rnn, 2017.

Jose Sotelo, Soroush Mehri, Kundan Kumar, Joao Felipe Santos, Kyle Kastner, Aaron Courville, and Yoshua Bengio. Char2Wav: End-to-end speech synthesis. In *ICLR Workshops*, 2017.

Stanley Smith Stevens, John Volkmann, and Edwin B Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 1937.

Andreas Stolcke. Srilm-an extensible language modeling toolkit. In *INTERSPEECH*, 2002.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *INTERSPEECH*, 2012.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *ICLR*, 2016.

Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. In *ICLR*, 2017.

Peter M Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 1989.

Keiichi Tokuda, Yoshihiko Nankaku, Tomoki Toda, Heiga Zen, Junichi Yamagishi, and Keiichiro Oura. Speech synthesis based on hidden Markov models. *Proceedings of the IEEE*, 2013.

Sandra E Trehub, Dale Bull, and Leigh A Thorpe. Infants' perception of melodies: The role of melodic contour. *Child development*, 1984.

Karen Ullrich, Jan Schlüter, and Thomas Grill. Boundary detection in music structure analysis using convolutional neural networks. In *ISMIR*, 2014.

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

Aäron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C Cobo, Florian Stimberg, et al. Parallel WaveNet: Fast high-fidelity speech synthesis. In *ICML*, 2017a.

Aaron van den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *NIPS*, 2017b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, et al. Tacotron: Towards end-to-end speech synthesis. In *INTERSPEECH*, 2017.

Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv:1804.03209*, 2018.

Ryuichi Yamamoto. WaveNet vocoder. https://github.com/r9y9/wavenet_vocoder, 2018.

Yujia Yan, Ethan Lustig, Joseph VanderStel, and Zhiyao Duan. Part-invariant model for music generation and harmonization. In *ISMIR*, 2018.

Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. MidiNet: A convolutional generative adversarial network for symbolic-domain music generation. In *ISMIR*, 2017.

Takayoshi Yoshimura. *Simultaneous modeling of phonetic and prosodic parameters, and characteristic conversion for HMM-based text-to-speech systems*. PhD thesis, Nagoya Institute of Technology, 2002.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv:1409.2329*, 2014.

Heiga Zen, Keiichi Tokuda, and Alan W Black. Statistical parametric speech synthesis. *Speech Communication*, 2009.

Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *ECCV*, 2016.