Data Hallucination, Falsification and Validation using Generative Models and Formal Methods

by

José Rafael Valle Gomes da Costa

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Machine Listening and Improvisation

and the Designated Emphasis

in

Computational and Data Science and Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Edmund Campion, Chair
Professor Sanjit Seshia, Co-Chair
Professor Ken Ueno

Spring 2018

Data Hallucination, Falsification and Validation using Generative Models and Formal Methods

José Rafael Valle Gomes da Costa

Abstract

Data Hallucination, Falsification and Validation using Generative Models and Formal Methods

by

José Rafael Valle Gomes da Costa
Doctor of Philosophy in Machine Listening and Improvisation
and Designated Emphasis in Computational and Data Science and Engineering

University of California, Berkeley
Professor Edmund Campion, Chair
Professor Sanjit Seshia, Co-Chair

The increasing pervasiveness and fast-paced development of deep learning (DL) systems with human-like perception, agency and creativity has brought concern related to information reliability: the generative models that have surprised and confused humans with their high quality media hallucinations can be used to fool other computer systems and humans to believe that the generated media is real. In addition to developing strategies to increase the quality of the data produced with generative models, specially generative adversarial networks (GANs), our research community has been exploring mechanisms to better understand and control the data they generate.

In the context of data falsification, formal methods (FM) and formal specifications (FS) can be used to prevent adversarial attacks by verifying that some potentially adversarial data follows the specifications of the real data. Formal methods and formal specifications can also be used to guide the output of a generator such that, as much as possible, the generated data fulfills the specifications of the real data. Although formal methods have been largely and sucessfuly used in the field of software and hardware engineering, their interaction with Artificial Neural Networks (ANNs) poses many challenges that are yet to be solved.

In this thesis, we address challenges related to hallucination, falsification and validation of data using generative models and formal methods. We start by focusing on artistic applications related to music by developing an automata-based system for machine improvisation with formal specifications. We briefly describe the Control Improvisation framework and describe its application to machine improvisation with formal specifications. We propose data abstractions derived from symbolic music data and describe strategies for mining specifications from them. We then use the mined specification to summarize a musical style and guide the improvisation of a generative model.

Next, we focus on security applications in speech synthesis and investigate the efficiency of generative models such as WaveNet, SampleRNN and our own GAN

model in performing spoofing attacks to fool a text-independent speaker recognition system. Inspired by universal background models (UBMs) in speech, we propose a modification to the Wasserstein GAN objective function such that data from multiple speakers can be used to generate data from a single speakers, increasing the efficiency of our spoofing attacks.

Last, in the hope of preventing adversarial attacks produced with data synthesized by generative models, specially GANs, we look at the properties of synthesized samples hoping to find traces of the data generation process that can be used to identify the data as adversarial. We empirically show that the data produced with these generative models do not follow the specifications of the real data and that they have universal properties, byproducts from the models and the algorithms used to train them, that can be used to identify the source of the data.

To my mom and dad.

Contents

List of Figures

List of Tables

Acknowledgments

I'm extremelly thankful to every single entity who directly or indirectly made this possible.

Chapter 1

Introduction

The desire to have statistical models with human-like perception and creativity has motivated our research community to develop strategies that increase the quality of the data produced with them, to explore mechanisms that prevent adversarial attacks they empower, and to better understand and control the data they produce. Within this context, this thesis addresses challenges related to hallucination, falsification and validation of data using generative models and formal methods.

In this section we motivate this thesis and situate it with respect to generative algorithms (GAs), artificial neural networks (ANNs), formal methods and the possible interactions therein. Formal methods is a field of computer science and engineering that is concerned with the the rigorous mathematical specification, design, and verification of systems [119]. We briefly describe the interface with GAs and ANNs and pose several research challenges related to these interactions, including challenges that can be addressed with formal methods. We focus on challenges related to the production and prevention of adversarial attacks with fake data, data generation with formal guarantees and understanding the properties of samples produced with generative adversarial networks (GANs). We then describe the organization of this thesis and the contributions offered therein.

1.1 Motivation

The increasing pervasiveness and fast-paced development of computer systems with human-like perception, agency and creativity has been dubbed by the 21st century media with the "clickbaity" name *Artificial Intelligence Revolution*. The Artificial Intelligence (AI) revolution has produced systems that can arguably perform tasks such as autonomous driving [16,106], free-form and open-ended visual-question answering [4, 135], and writing romantic symphonies under the supervision of a human [10,30]. Preceded by a period in the 80s that has been called the AI winter [62], the AI revolution and renaissance became possible through recent advancements in parallel

and distributed systems [11,32] that enabled feeding huge amounts of data [1] into deep learning models, a subset of machine learning that uses artificial neural networks (ANN) to train mathematical models successful in classifying and generating data.

Surprisingly, our extreme success in the use of ANNs to solve tasks that require human-like intelligence is not accompanied by a clear understanding of ANNs in general. Our lack of understanding is such that in 2014 the field was surprised with the famous example [56] in which a ANN-based object recognition model classifies an imperceptibly modified image of a panda as a gibbon with 99% confidence. This limited understanding of such systems, their increasing pervasiveness and the amount of personal data collected with them raises several concerns. What are the guarantees provided by such systems [127]? How robust are they to adversarial attacks [55]? Are these systems so powerful that they can be used to generate fake data that can fool humans and machines [79]? These are a few challenges that must be addressed towards reaching the goal of verified artificial intelligence [120].

Artificial intelligence and computer science have also strongly impacted art to the extent of enabling the development of new styles such as Spectral music [44] and Inceptionism [97], a new style of visual art. The first published use of algorithmic music composition [90] dates back to the Illiac Suite of the 1950's. The limited understanding and control over the output of generative algorithms has impacted art in different ways: while some embraced it as a blessing to be incorporated into their art form, the community interested in controlled machine improvisation has seen the lack of formal guarantees as a curse and challenge that must be addressed.

1.2 Interfacing with Generative Algorithms

We can define generative algorithms as the set of non-deterministic algorithms that generate data in some domain given some input or *seed*. In the sonic arts a famous example is Mozart's Musikalisches Wuerfelspiel and in the visual arts Google's DeepDream [97].

These algorithms can be a sequence of instructions, such as the flow-chart score to Michael Philippot's *Composition for Double Orchestra* [137], or a mathematical model with which one can generate new data given some context, such as Deepak Pathak's Context-Encoder [104]. There are also systems that have a mixed approach in which mathematical models are combined with instructions and rules such as David Cope's Experiments in Musical Intelligence (EMI) [29], George Lewis' Voyager [80] and Edmund Campion's Nat-Sel [87].

Lately, the production and consumption of data produced with generative algorithms has been increasing and finding success in data augmentation [122], adversarial attacks [121,136], music composition [10] (officially recognized by SACEM as a composer), speech synthesis [86] (a Montreal startup that claims it can recreate any voice

[1] *Data is the new oil.*, as stated by Clive Humby in 2006.

using just one minute of sample audio), and art venues such as the successful auction of images generated with AI [134].

1.2.1 Rule-based and data-driven approaches

Generative algorithms are normally designed using rule-based and data-driven approaches [115]. Rule-based approaches attempt to define rules characterizing the expected behavior or rules of a system generating data in some domain. Rule-based approaches have the advantage that the rules of the system are written by a domain specialist and, hence, are interpretable. For example, a musicologist can describe rules that can be used to write jazz melodies [69]. On the other hand, data-driven approaches learns from the data the parameters of the model that describes the joint distribution of input and output [14]. For example, one can collect jazz licks and try to use data to train a model that is capable of producing jazz licks [50]. Naturally, it is possible to combine both approaches [93].

It has been observed, specially in music, that it is difficult to come up with the proper rules, resulting in systems that are either too restrictive, limiting creativity, or too relaxed, thereby allowing undesirable behavior [29,37,70]. Constrained on the data available and compared to rule-based approaches, data-driven have better generalization, do not require the potentially cumbersome task of writing the rules that describe the generation process [2] and can exploit latent properties in the data that can be unknown to specialists or even the producer of the data.

1.3 Interfacing with Artificial Neural Networks

ANNs are extremely efficient mathematical objects capable of learning functions that can be used to learn a map from an input to an expected output. Work based on the universal approximation theorem [31,65] claims that a multi-layered feed-forward network with limited capacity can approximate continuous function on compact subsets of $\mathbb{R}^n$. ANNs are so powerful that they can fit random noise with 100% accuracy, as described in [143].

1.3.1 Interpretability

The interpretation of neural networks is an on-going research question that has been been approached from data and network-centric perspectives [140]. Data-centric approaches can, for example, look for individual neurons [3] or groups of neurons that are highly activated given some properties in an image [142], e.g. the "cat neuron". Notably, theoretical work in [127] claims that in the high layers it is the

[2] Imagine writing rules that describe how to draw Surrealist art.

[3] Neuron in the sense of neural network units.

space, rather than the individual neurons, that contains the semantic information. An existing network-centric approach uses the gradient of ANNs to find images that cause higher [123] or lower [127] activations for output units. The same network-centric approach can be used to attack an ANNs by finding the smallest modification to an image that makes the ANN output a label that is not the label associated with the image. This network-centric approach is one of the pillars of adversarial attacks [56,96] to classifiers that are built using ANNs.

1.3.2 Robustness to adversarial attacks

Our limited understanding of ANNs also has consequences on their robustness to adversarial attacks. Whereas there is a belief that that ANN's non-robustness to some adversarial attacks is due to their non-linear nature, [56,127] argue that ANNs learn input-output mappings that are fairly discontinuous and that the main cause to the vulnerability of ANNs to adversarial attacks is their linear nature. This creates an interesting paradox in which models that are easier to train due to their linearity become more prone to adversarial attacks.

1.4 Research Challenges

The interface with generative algorithms and neural networks has interesting research challenges. Broadly speaking, current research trends in generative models are focused in increasing the quality of generated samples [79,88,103] or enforcing specifications on generated data [1,36,46], estimating what is their efficiency in fooling humans and machines [121,136], understanding what are the properties of the samples produced by them [8,9,128]. A formal specification is a mathematical statement of what a system must or must not do, often expressed in mathematical logic or using automata-theoretic formalisms.

1.4.1 Data generation for targeted and untargeted attacks

The rise of generative models producing high quality audio [92,130], video [125], image [79] and text [57,144] data raises questions related to security and information reliability. Generative models have been used to produce fake data that has successfully fooled both humans and machines [56,121,136]. As a matter of fact, the topic is so relevant that in 2017 NIPS created a competition track named *Adversarial Attacks and Defenses* [126] and The Economist magazine has produced two pieces [40,41] wherein they discuss generative models and their ability to hallucinate [4] fake but convincing audio and video data.

[4] Hallucination is the term used by the deep learning community to refer to surreal images produced with deep learning models.

1.4.2 Data generation with formal guarantees

Generative models can be coupled with controllers that restrict their behavior such that, as much as possible, the data generated fulfills the specifications [109]. Informally speaking, specifications are deterministic or non-deterministic rules that describe the expected behavior of a system [26]. In music, for example, there are several rules for counterpoint [24], harmony [110], that describe the expected behavior of a musical style. Although there has been work on combining Neural Networks and Control [94], there are many challenges related to verified artificial intelligence that are yet to be solved [120].

While generative models using ANNs are extremely powerful, it is not trivial to predict the outcome given the input, specially in situations where the output of the system is recursively used as input. On the other hand, models based on automata are more predictable but usually lack the power and flexibility of ANNs. The Control Improvisation [45,46] framework addresses these issues by combining data-driven learning and controller synthesis from formal specifications.

1.4.3 Properties of GANs samples

The abundance of papers on GANs [63] and the high quality image samples produced with them is inversely proportional to how much we understand their properties. A few researchers, specially Sanjeev Arora and have been raising questions about what exactly GANs learn and what are the properties of the samples they generate. There are empirical and theoretical studies in [8,9] showing that "current GANs approaches, specifically, the ones that produce images of higher visual quality, fall significantly short of learning the target distribution, and in fact the support size of the generated distribution is rather low". Very interesting work [128] on the evaluating of generative models emphasizes that visual fidelity is a bad predictor of true log-likelihood performance and that even a simple *k-means* based approach can obtain better Parzen windows performance than using the original samples from the dataset, even though they come from the exact same distribution!

1.5 Thesis Organization and Contribution

In this thesis, we address challenges related to hallucination, falsification and validation of data using generative models and formal methods. We start by focusing on artistic applications related to music by developing an automata-based system for machine improvisation with formal specifications.

Next, we focus on security applications related to speech and investigate the design of neural network generative models for spoofing attacks to speaker recognition systems. These models are later applied to music applications. Last, in the hope of preventing spoofing attacks, we look at the properties of generated samples with

GANs hoping to find traces of the data generation process that can be used to identify the data as being adversarial.

1.5.1 Specification Mining for Machine Improvisation with Formal Specifications

Chapter 3 tackles the problem of mining specifications from symbolic music data and using the specifications in a machine improvisation system for music with formal specifications. We propose an algorithm for mining specifications from symbolic music data using pattern graphs, introduce the concept of hard and soft specifications and show results comparing machine improvisations with and without specifications. This work was done jointly with the UC Berkeley's Control Improvisation project led by Professors Sanjit A. Seshia Professor, David Wessel, and Professor Edward Lee and including Adrian Freed, Alexandre Donzé, Ilge Akkaya, Daniel Fremont, Sophie Libkind.

1.5.2 Adversarial Attacks to Speaker Recognition Systems using GANs

Chapter 4 analyzes the efficiency of generative models in fooling text-independent speaker recognition systems through targeted and untargeted attacks. Within this task, we investigate the performance of state-of-the-art generative models for speech and our own GAN based generative model. We propose a modification to the objective function of the Wasserstein GAN with gradient penalty to enable semi-supervised training, where we train one GAN per target speaker using real data from the target speaker and other speakers. This work was done jointly with Anish Doshi and Wilson Cai.

1.5.3 Interesting properties of GAN samples

In Chapter 5 we analyze properties of GAN samples from different sources, including Handwritten Digits, Music and Speech. We speculate that the generated samples have an universal property that is dependent on the learning setup, i.e. stochastic gradient descent and the non-linearities used. This universal property and other factors impact the difference between the distribution of features computed over the real data and the fake generated data. We also show that samples produced with GANs can violate specifications of the real data distribution, although to the bare eyes they look similar to the real data. This work was done jointly with Anish Doshi and Wilson Cai.

Chapter 2

Preliminaries

The research in this thesis relies heavily on automata and artificial neural networks. In this section, we superficially describe them with the purpose of arming the reader with good intuition on these subjects. Whenever necessary, we provide references to publications that describe these models and aspects thereof in depth.

2.1 Automata

The word automaton comes from ancient Greek and refers to self(autos)-acting entities. In computer science, automata theory deals with the definitions and properties of mathematical models of computation. The simplest model of computation possible is the finite state machine or finite automaton (FSA). In the following paragraphs we provide some basic intuition and formal definitions therein.

Let's start with a simple automaton example: an automatic faucet. The automatic faucet is machine with a gate that controls water output based on human presence. The gate has two states, open or closed, and faucet's controller has a single bit of memory that records in which state the faucet is. The gate transitions between these two states conditional on human presence. For example, if the gate is open and a there is no human presence, the gate will transition from open to closed. Describing a complex FSA in such a manner will be complicated and require several lines of text. An efficient alternative is to represent automata with state diagrams:

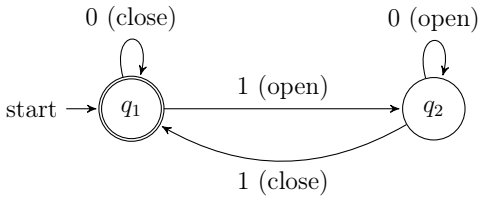


Figure 2.1: State diagram of the hypothesized automatic faucet.

Figure 2.1 depicts a state machine where the **states** are represented with labeled circles and transition are represented with labeled **arrows**. The **start state** q_1 is identified by the arrow pointing at it from nowhere and it represents a state from which the machine should start its execution. The **accept state** is identified by the double circle and represents a state in which the machine can finish its execution.

Perhaps unexpectedly, automata can be used to describe the specifications or desired behavior of some traditional western music. For example, the **start** and **accept** states can be interpreted as valid notes to start and end a melody and the **transitions** as how these notes can be reached. In Chapter 3, we address this subject in detail, describing how to learn specifications from data and how to use them to control musical agent that improvises.

2.1.1 Deterministic and Non-Deterministic Finite Automata

Now that we have broadly described the concept of an automaton, we will focus on formal definitions, starting with the deterministic finite automaton (DFA).

Definition 1 (Deterministic Finite Automaton)**.** *A Deterministic (DFA) finite state automaton is a 5-tuple* $\mathcal{A} = (Q, q_0, F, \Sigma, \rightarrow)$ *where* Q *is a finite set of* states*,* $q_0 \in Q$ *is the* initial state*,* $F \subset Q$ *is the finite set of* accepting states[1]*,* Σ *is a finite set called the* alphabet *and* $\rightarrow \subset Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ *is the* transition relation *trans*$(\mathcal{A})$*.*

The execution of an automaton $\mathcal{A}$ operates such that given a state, possibly the current state of the automaton, and an input symbol representing a transition, the transition function returns a state:

Definition 2 (Execution)**.** *An execution of an automaton* $\mathcal{A}$ *is a sequence* $q_0\sigma_0q_1\sigma_1...$ *where* $(q_i, \sigma_i, q_{i+1}) \in trans(\mathcal{A})$*. Executions can be finite or infinite and must end in a state.*

Let's consider the letters of the alphabet as observable events of the execution of the automata under consideration, for example the state of the faucet automaton. A *word* w is either ϵ (transition with no observable event) or a finite sequence of letters in Σ, i.e. $w = \sigma_1\sigma_2 \dots \sigma_k$ for some integer $k \geq 1$. The length of a word is defined inductively as $|\epsilon| = 0$ and $|w\sigma| = |w| + 1 \ \forall\sigma \in \Sigma$.

To illustrate, let's use the letters c and o to represent states q_1 and q_2 from our faucet automaton and evaluate the execution of the input string *1*. With this input string, the machine starts on q_1, transitions from q_1 to q_2 by means of 1. This can be summarized as $q_1 \xrightarrow{1} q_2$ and produces the output word $w = co$. This leads us to the definition of traces and accepting traces:

[1] Also known as final states.

Definition 3 (Trace)**.** *A trace of $\mathcal{A}$ is a record of some sequence of instructions executed by $\mathcal{A}$. A word is a* trace *of a FSA $\mathcal{A} \iff$ there exists a sequence of states $q_i \in Q$ such that $q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_{n-1}} q_{n-1} \xrightarrow{\sigma_n} q_n$. It is an* accepting trace *of $\mathcal{A} \iff q_n \in F$. The* language of $\mathcal{A}$*, noted $\mathcal{L}(\mathcal{A})$ is the set of accepting traces of $\mathcal{A}$.*

Although finite state automaton can be extremely useful, they have limitations when taking a single transition can lead to multiple states. In such cases Non-Deterministic Automata can be used:

Definition 4 (Non-Deterministic Finite Automaton)**.** *A Non-Deterministic (NFA) finite state automaton is a 5-tuple $\mathcal{A} = (Q, q_0, F, \Sigma, \rightarrow)$ where Q is a finite set of* states*, $q_0 \in Q$ is the* initial state*, $F \subset Q$ is the set of* accepting states*, Σ is a finite set called the* alphabet *and $\rightarrow \subset Q \times (\Sigma \cup \{\varepsilon\}) \times P(Q)$ is the* transition relation*, where $P(Q)$ is the **power set** of Q..*

The difference between DFAs and NFAs resides on the transition function. Whereas DFA's transition function takes a state and an input symbol and outputs a *state*, NFA's transition function also takes a state and a symbol *or* the empty string ε and outputs *the set of possible next states* [124].

2.1.2 Formal specification

In the previous subsection, we briefly mentioned that automata can be related to specifications in music. A formal specification is a mathematical statement, described in mathematical logic or automata-theoretic formalism, that defines the expected behavior of a system. Formal specifications are central to certain fields of computer science, such as program verification or supervisory control. Supervisory control refers to the problem of designing a **controller** (supervisor) that guarantees that a **system** (plant) always satisfies a set of formal specifications. If we think of formal specifications as music rules and the "plant" as a random improviser, then the supervisory controller is similar to a controlled improviser of music. In supervisory control, the combination of a **system** and a **controller** can be done via a synchronous product:

Definition 5. *(Synchronous Product) The synchronous product of $\mathcal{A} = (Q, q_0, F, \Sigma, \rightarrow)$ and $\mathcal{A}' = (Q', q_0', F', \Sigma, \rightarrow')$, noted $\mathcal{A}||\mathcal{A}'$, is defined as the FSA $\mathcal{A}||\mathcal{A}' \triangleq (Q \times Q', (q_0, q_0'), F \times F', \Sigma, \rightarrow)$ where $\forall \sigma \in \Sigma \cup \{\varepsilon\}$, $(q_i, q_i') \xrightarrow{\sigma} (q_j, q_j')$ if and only $q_i \xrightarrow{\sigma} q_j$ and $q_i' \xrightarrow{\sigma}' q_j'$.*

Intuitively, $\mathcal{A}||\mathcal{A}'$ is a finite state machine where $\mathcal{A}$ and $\mathcal{A}'$ take transitions synchronously, with the constraint that for a transition to be possible, both current states of $\mathcal{A}$ and $\mathcal{A}'$ must have an outgoing transition driven by the same event. In Chapter 3 we further develop this topic and describe our research related to mining specifications from symbolic music data and music improvisation using the control improvisation framework [45,46].

2.2 Artificial Neural Networks

The other class of models used in this thesis are Artificial Neural Networks (ANNs), also know as Deep Neural Networks or Deep Learning. The name informally draws inspiration from theoretical models of how learning could happen in the brain. Deep learning is not new and according to [54], there have been three eras: *cybernetics* in the 1940s-1960s, *connectionism* in the 1980s-1990s and the current Deep Learning renaissance beginning around 2006. Mathematically speaking, a fully-connected neural network is a chain of equations, possibly non-linear, whose parameters are estimated using methods such as stochastic gradient descent and backpropagation.

We introduce ANNs through linear and logistic regression and use Figure 2.2 as a reference. Linear regression is used to estimate the parameters that model the relationship between an output variable given input variables. Mathematically, it can be described as a weighted sum of inputs of the form[2]:

$$z = \mathbf{w} \cdot \mathbf{x} + b \tag{2.1}$$

where weights $\mathbf{w}$ and inputs $\mathbf{x}$ are vectors in $\mathbb{R}^n$, b is a scalar bias term, and z is a scalar. The output of single neuron without non-linearities is similar to the output of the linear regression described previously. Logistic regression is a special version of regression where a specific non-linear function, i.e. the sigmoid function, is applied to the output of the linear regression described in Equation 2.1:

$$g(z) = \frac{1}{1 + \exp^{-(z)}} \tag{2.2}$$

This non-linear equation is similar to the output of one neuron with a sigmoid non-linearity, also known as activation function. A combination of such neurons, with their respective non-linearieties and weights, defines a simple layer in a neural network, and a chain of layers [3] defines a simple form of neural network. The output of a hidden layer is described by Equation 2.3 and Figure 2.2 below:

$$\mathbf{h}^{(l)} = g^{(l)}(\mathbf{w}^{(l)} \cdot \mathbf{x} + b^{(l)}) \tag{2.3}$$

Equation 2.3 illustrates the similarity between linear/logistic regression and a simple fully connected ANNs. We use Figure 2.2 to illustrate Equation 2.3 and the flow of data in a simple fully connected ANNs.

For more details and a thorough explanation of deep learning, we forward the reader to the book [54].

[2] We ignore the error term.

[3] Lasagne [33] is a well know Italian dish and deep learning library

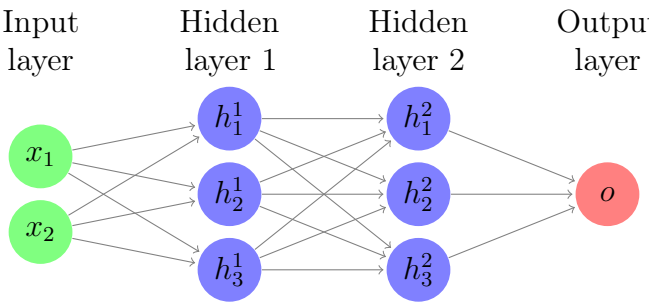


Figure 2.2: Graphical representation of a fully conected neural network with two inputs, two hidden layers with three nodes each and one output node. The weights are represented by arrows and the activation function is implicit

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [78] are neural networks that convolve filters or kernels, i.e. tensors in $\mathbb{R}^d$, that are convolved with the data to produce output. These parameters of these filters are estimated during training and in the image domain they are square with small sizes ranging from 3x3 to 9x9 in pixel size. The convolution operation can be interpreted as sliding a filter over the data and for each reached location applying a dot product between the filter and the data at the location. Figure 2.3 below shows a simple example.

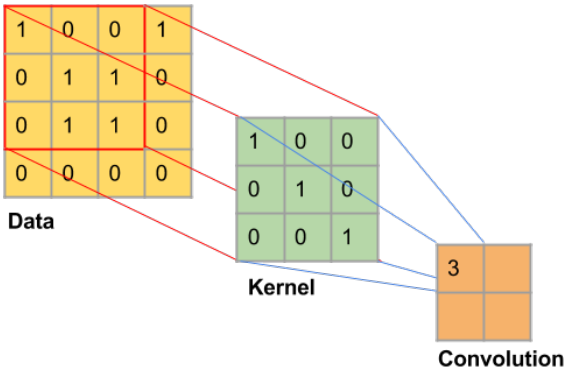


Figure 2.3: An intermediary step of convolution with stride 1. The kernel in green is convolved with the first area in the data, represented by the red square and the output is stored on the first cell of the orange grid.

A special characteristic of CNNs is that the weights of the filters are learned. For example, if the task at hand is classifying handwritten digits from the MNIST dataset, the ANN might learn filters that look like vertical, horizontal and diagonal lines [140].

For more information on CNNs and convolution arithmetic, we again forward the reader to [54] and the excellent *A guide to convolution arithmetic for deep learning* [39].

2.2.2 Backpropagation

The back-propagation, or backpropagation, algorithm [112] is a special case of reverse mode automatic differentiation [83]. In its basic modern version [83,114], the back-propagation algorithm has become the standard for training neural networks, possibly given its underlying simplicity and relative power [113]. Inspired by the work of Donal Hebb and the so-called Hebb rule [61], Rosenblatt developed the idea of a Perceptron that was based on the formation and change of synapses between neurons, where the output of a neuron would be modeled as a weighted sum of its incoming signals. This is similar to what we described in Equation 2.3.

The basic idea of estimating the parameters for a model with back-propagation is as follows: define an error function, compute the gradient of the loss with respect to the weights using back-propagation and perform gradient descent to find a weight update that is optimal for minimizing the error given the current data. From a perspective of calculus, back-propagation is an algorithm that efficiently computes the chain rule, with a specific order of operations [54].

The data and the error function are extremely important aspects of the learning procedure that will be used to estimate the parameters of the model. For example, let's consider training a classifier to identify handwritten single digits. Let's assume we decide to train the model using the famous MNIST [77] dataset, containing 28 by 28 monochromatic images of single handwritten white digits on a black ground. Ideally, we want the model to predict 1 whenever the image looks like a 1, 2 whenever it looks like a 2 and so on. Back-propagation does not define what aspects of the images should be considered nor does it provide guarantees that the classifier will generalize to unseen examples and will not simply memorize the data used during training. Indeed, a model trained with the data we described will considerably fail if the colors are inverted, i. e. black digits on white background. Note that, in this case, regularization methods that do not rely on data augmentation will probably also fail in helping the model generalize to other color schemes. One common procedure to fix such issues is to augment the data and include images with all combinations of foreground and background color [4].

2.2.3 Tricks of the Trade

Neural Networks: Tricks of the Trade is the title of the book released by Springer in 2003 [100] wherein several authors provided practical advice on training neural networks. This section addresses some of such tricks, including tricks available in [13,76], that were relevant to the execution of our research.

[4] Certainly not the most elegant nor abstract solution to the problem of generalization.

2.2.3.1 Input Normalization

Input normalization is an old trick, if not requirement, for training neural networks. Data with very large of very small numbers can lead to exploding or vanishing gradients, respectively. Therefore, a common pre-processing step in training neural networks is to standardize the data. Following normative practice, the mean and standard deviation computed on the training data are used to standardize any new data, including validation and test data. Input normalization is a costless and efficient procedure that contributes to weight normalization.

2.2.3.2 Weight Normalization

In neural networks weight normalization can prevent overflow and underflow of gradients [51,60,73] and to avoid covariate shift. In [66], the authors explain that the covariate shift can cause the input distribution not to be fixed, possibly leading to exploding and vanishing gradients. Batch normalization is a remedy that does not come for free because it requires estimating additional parameters.

Weight-norm [117] is another efficient and recent weight normalization technique It is a reparameterization of the weight vectors in a neural network that decouples the length of those weight vectors from their direction.

Very recently, the Soft Exponential Linear Unit (SELU) [72] has been introduced under the premise that it can be more efficient than weight normalization techniques in fully connected, recurrent and convolutional neural networks. Weight normalization can be a remedy for poor parameter initialization.

2.2.3.3 Parameter Initialization

Faster convergence can be achieved with parameter intialization and, in the case of GANs, convergence itself might only happen with proper parameter initialization. Currently, the main strategies for parameter initialization are described in [51,60,72,73] and focus on initializing weights with a normal distribution with zero mean and variance dependent on the number of incoming and outgoing units and the non-linearity at hand.

Proper parameter initialization is extremely important for gradient propagation and poor initialization might impede the network from converging, e.g. Andrej Karpathy, director of AI at Tesla, was not able to reproduce experiments on a 20-layer network by initializing network weights with $N(0, 0.02)$ but only $N(0, 0.05)$.

2.2.3.4 Non-Linearities

Traditionally, the sigmoid non-linearity dominated the field of ANNs but it has been replaced by other non-linearities such as tanh and the rectified linear unit(relu) [60]. The gradient of the sigmoid non-linearity quickly and asymptotically

approaches zero as the magnitude of the input increases, providing only a small gradient for most of its domain.

The hyperbolic tangent function shows a similar behavior to the sigmoi function but with gradients that are considerably stronger in the vicinity of 0.

Although the rectified linear unit, a.k.a. half-wave rectifier, has 0 gradient in half its domain, it has been replacing the sigmoid and tanh non-linearities due to it's computational efficiency and tendency to produce sparse representations.

The recent and promising selu [72] non-linearity is believed to produce self-normalizing networks and provides evidence of doing so on fully connected, convolutional and recursive neural networks.

2.2.3.5 Regularization

It is rather common that neural networks have more parameters than data points being learned, thus raising questions regarding overfitting the training data. Recent research shows that deep neural networks with enough capacity are capable of learning a classifier trained on data with random labels [143], even if regularization is used to circumvent the problem of overfitting.

Similar to regression models, the L1 and L2 norm are regularization techniques that are used in neural networks. Whereas both L1 and L2 regularization penalize model complexity, L1 also drives sparsity.

Another technique, or heuristic, is believed to prevent overfitting is early stopping [98], where a criteria based on performance degradation on the validation is used to stop training.

Dropout [64] is another easy to implement regularization technique where for each iteration the nodes of the network are set to zero with probability p. During inference, the learned weights are scaled by $1/p$.

Finally, data augmentation is another procedure related to regularization that is very efficient when it is feasible to augment the data. It operates under the assumption that regularization is not necessary if the data available is representative of the real data distribution. Compared to other regularization techniques, data augmentation has the benefit of possibly training models that are invariant to the transformations applied in data augmentation, e.g. invariant to rotation and scale.

2.3 Generative Adversarial Networks

Generative Adversarial Networks [55](GANs) hare evolving rather quickly and have been receiving a considerable amount of attention recently, including Yann LeCun's comment expressing that the GAN framework is one of the most important topics in deep learning.

The framework has been applied to many tasks, including unsupervised feature learning [104,108], text to image synthesis [111], image super-resolution [79], symbolic music [139] and many others.

In addition to being applied to many tasks, many variations of the GAN framework have been proposed, including Energy-Based GANs [145], Boundary Equilibrium GANs [15], Mix-GANs [8], Least-Squares GANs [88], Wasserstein GAN [7] and finally[5] Wasserstein GAN with Gradient Penalty [59].

2.3.1 Adversarial Neural Networks

Adversarial Neural Networks were first described in the landmark paper [55] in 2014. The setup of the framework uses an adversarial process to estimate the parameters of generative models by iteratively and concomitantly training a discriminator network D and a generator network G. One of the main advantages of GANs is that, unlike other deep generative models that use approximation methods to compute intractable functions or inference, GANs do not require an approximation method.

Informally speaking, the discriminator network plays the role of an investigator that learns to distinguish between samples that are *real*, i. e. samples that come from the distribution generating the training data, and samples that are fake, i. e. samples produced by the generator. The generator network plays the role of a counterfeiter that uses feedback from the discriminator to learn how to produce samples that are capable of fooling the discriminator. This informal description already calls attention to issues that might be present when using the framework:

- A week investigator might be easily fooled by the generator.
- An investigator without capacity might not learn to distinguish the data properly.
- An investigator that disregards variety can be fooled with a single example.

Formally speaking, D is a function $f : x \mapsto y$, where x is some input sample in $\mathbb{R}^d$, e.g. an image, and y is a scalar, e.g. representing D's predicted probability that x comes from the real distribution. G is a function $f : z \mapsto \tilde{x}$, where z is a noise vector in $\mathbb{R}^d$ and $G(z) = \tilde{x} \in \mathbb{R}^d$ is an image produced by the generator. D and G play a two-player minimax game with the value function V(G, D):

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}}[logD(x)] + \mathbb{E}_{z\sim p_z}[log(1 - D(G(z)))] \tag{2.4}$$

One of the most important assumptions described in the seminal GAN paper is that the model will converge to the true distribution if the discriminator and generator have infinite capacity and there is a large amount of data. Work done in [8,9] poses questions about the assumption and evaluates networks using the birthday paradox

[5]The field is moving absurdly fast and this list will be outdated at the time of release.

test. Although this test is extremely dependent on the distance measure obtained to identify similar objects, it is an upper bound approximation! Furthermore, work in [128] shed theoretical and empirical light on this assumption: they describe that there is considerable independence between sample quality and likelihood, showing that high likelihood and sample quality does not necessarily imply a good model given independence between pixels. Naturally, in natural images neighboring pixels are normally not independent.

The objective function described in Equation 2.4 is equivalent to minimizing the Jensen-Shannon (JS) divergence between the distributions, as described in [55]. The Jensen-Shannon divergence is a symmetric and smoothed version of the Kullback-Leibler (KL) divergence [75].

$$D_{JS}(P \mid\mid Q) = \frac{1}{2} D_{KL}(P \mid\mid M) + \frac{1}{2} D_{KL}(Q \mid\mid M), \tag{2.5}$$

where D_{KL} is defined as :

$$D_{KL}(P \mid\mid Q) = \int_{-\infty}^{+\infty} p(x) \log \frac{q(x)}{p(x)} dx \tag{2.6}$$

It can be seen from Equation 2.6 that KL based objective functions suffer from exploding loss when the support of the real distribution P is not contained on the other distribution Q, that is, the KL divergence goes to $-\inf$ if there is some x such that $Q(x) = 0$ where $P(x) > 0$. In addition, the authors in [7] provide a thorough comparison of different distances and explain that there are distributions where JS, KL and even Total Variation divergence do not converge and have gradients always equal to 0.

Another problem with the original GAN objective function in Equation 2.4 is that the sigmoid function saturates quickly and, for this reason, will barely consider the distance of a sample to the decision boundary formed with the sigmoid[88].

These problems associated with the original GAN objective function have been addressed with the development of Least-Squares GAN(LSGAN) [88] and Wasserstein GAN(WGAN) [7].

2.3.2 Least Squares GAN

The *Least Squares GAN* [88] uses the least squares objective function to train the discriminator and generator. Unlike the sigmoid cross entropy, the least squares loss more heavily penalizes samples regarding their position with respect to the decision boundary. In their paper [88], the authors affirm that the LSGAN contributes to the stability of the learning process, removes the need of using Batch Normalization and converges faster than the Wasserstein GAN.

2.3.3 Wasserstein GAN

The *Wasserstein GAN* [7] (WGAN) framework instead uses the Wasserstein (Earth-Mover) distance between distributions, which in many cases does not suffer from loss explosion and vanishing gradient. In the WGAN framework, the loss functions of the generator and *critic* (which no longer emits a simple probability, but rather an approximation of the Wasserstein distance between the fake and real distributions) become:

$$L_G = -\mathop{\mathbb{E}}_{\widetilde{\boldsymbol{x}}\sim\mathbb{P}_g}\big[D(\widetilde{\boldsymbol{x}})\big] \tag{2.7}$$

$$L_C = \mathop{\mathbb{E}}_{\widetilde{\boldsymbol{x}}\sim\mathbb{P}_g}\big[D(\widetilde{\boldsymbol{x}})\big] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big] \tag{2.8}$$

where P_r is the real distribution and P_g the distribution learned by the generator. The original WGAN framework uses weight clipping to ensure that the critic satisfies a Lipschitz condition.

2.3.4 Wasserstein GAN with Gradient Penalty

As pointed by [59], however, this weight clipping can lead to problems with gradient stability. Instead, [59] suggests adding a gradient penalty to the critic's loss function, which indirectly tries to constrain the original critic's gradient to have a norm close to 1. Interestingly, already in 2013, [127] proposed a method to direct neural networks to become k-Lipschitz by penalizing the objective function with the operator norm of the weights of each layer. Equation 2.8 thus becomes (taken from [59]):

$$L_C = \underbrace{\mathop{\mathbb{E}}_{\widetilde{\boldsymbol{x}}\sim\mathbb{P}_g}\big[D(\widetilde{\boldsymbol{x}})\big] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{x}}}\big[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\big]}_{\text{Gradient Penalty}} \tag{2.9}$$

2.3.5 Tricks of the Trade

Similar to tricks for training neural networks, there are a few sources [25,116] that provide best practices for training generative adversarial networks. These best practices were mainly developed to circumvent the difficulty in training GANs using the objective function described in 2.4. These tricks might not apply nor be necessary to other GAN formulations such as LSGAN or WGAN.

2.3.5.1 Objective Function

The first formulation of the objective function of the generator

$$\min_G \mathbb{E}_{z\sim p_z}[log(1 - D(G(z)))], \tag{2.10}$$

has vanishing gradients early on. In practice, it's preferable to use:

$$\max_G \mathbb{E}_{z\sim p_z}[\log D(G(z))] \tag{2.11}$$

2.3.5.2 Adding instability

It is reported that adding instability to the training procedure can improve training. Common strategies include occasionally flipping labels, i. e. real becomes fake and fake becomes real, adding time decaying uniform noise to the input data, and using label smoothing, that is, replace labels 0 and 1 with a random number close to 1 or 0 respectively. In our experiments in Chapter 4 we used time decaying uniform noise to prevent the discriminator from overfitting the data.

2.3.5.3 Optimizer

Given the unstable and unpredictable nature of GAN training, it is better to use adaptive learning methods such as ADAM or RMSProp. In our experiments with the MNIST dataset and GANs with different objective functions (GAN, LSGAN, WGAN, WGAN-GP) described in Chapter 5 we found that ADAM performs better than RMSProp, producing better samples earlier in the training process.

2.3.5.4 Learning rate

One of the challenges in training neural networks is setting the proper learning rate. Within the GAN framework wherein one looks for an equilibrium between the discriminator and the generator networks, this challenge gets exacerbated given the presence of a learning rate for each network and the dependencies between losses.

For discriminator and generator networks of equal size, one expects the discriminator to have some advantage over the generator such that the updates provided by the discriminator to the generator are useful. This can be achieved by setting the discriminator's learning rate to be slightly higher than the generator or by performing more discriminator updates than generator updates.

For discriminator and generator networks where the generator is larger than the discriminator, the strategy can differ. For example, let's consider a context-encoder generator as the one described in [104]. In this example, the generator has more parameters than the discriminator given that it has both an encoder that interprets the context and a decoder that generates the missing data. In this case, the training process of the generator is slower and the balance between discriminator and generator can be disrupted if the learning rates are the same. This setup was used in [104] and in our experiments on attacking speaker verification systems described in Chapter 4.

2.4 Concluding Remarks

Now that we have superficially explained the theoretical pillars of the research in this thesis, we will describe in depth the three projects that were summarized in subsection 1.5. These projects address data hallucination, falsification and validation in different domains using generative models and formal methods.

Chapter 3

Specification Mining for Machine Improvisation with Formal Specifications

3.1 Introduction and Motivation

The field of *machine improvisation*, i.e. computer music improvisation, has been investigated under mainly two approaches: rule-based and data-driven. Rule-based approaches attempt to define rules characterizing "good" improvisations and generate pieces of music that follow these rules. However, it has been observed that it is difficult to come up with the "right" rules, resulting in systems that are either too restrictive, limiting creativity, or too relaxed, thereby allowing undesirable behavior [29,37,70]. Data-driven approaches tend to employ machine learning techniques to learn generative models from music samples and use these models to generate new melodies. Examples of such models include stochastic context-free grammars (SCFGs) [49,69], hidden Markov models (HMMs) [50,102], and universal predictors [12,18,37,38]. Some systems combine rule-based and data-driven approaches; e.g. the Impro-visor system [70] based on SCFGs uses rules learned by grammatical inference from training licks [49]. Related to our work, [101] describe non-homogeneous Markov processes with control constraints, e.g. last pitch must be a specific note. While Pachet's work focuses on unary constraints manually created by the user and binary constraints that are within the scope of the Markov order, this research focuses on learning constraints from data as formal specifications. Our recent efforts in this direction are presented in [36], in which we define the problem of machine improvisation with formal specifications. In computer science, a *formal specification* is a mathematical statement of expected behavior of a system, typically given in mathematical logic or as an automaton. In [36], we considered the scenario of improvising a monophonic jazz melody given a training sequence (melody) and a chord progression. Overall, our approach described

in [36] consists of two stages: a generalization stage, where a reference sequence (e.g. obtained from a human improviser) is used to learn an automaton generating similar sequences, and a supervision stage, that enforces specifications on pitch and rhythm.

In [36], the specification, formally represented as a finite state automaton (FSA), encodes rhythmic and harmonic constraints adapted and simplified from generic jazz improvisation guidelines found in Keller's *How to Improvise Jazz Melodies* [69]. Although these hand-crafted guidelines can be manually converted into formal specifications, this task is time-consuming even in simple cases. Generally, writing specifications requires knowledge of logic not possessed by most composers and, conversely, musical knowledge not possessed by most logicians. *Specification mining* offers a solution that is either entirely automatic or only requires the much simpler task of creating templates for the musical patterns of interest. Our engine statistically learns, in the form of a pattern graph, the musical characteristics of a song dataset by mining predefined musical and general usage patterns from it. In this research, we evaluate our approach using a dataset of traditional blues songs and the predefined patterns focus on properties related to rhythm, pitch, melodic contour and chord/non-chord tones.

The chapter is organized as follows. Section 3.2 gives an overview of our approach, using a simplified presentation and a small example, followed by related work in Section 3.7. Section 3.3 describes the control improvisation and specification mining formalisms our techniques are based on, while the algorithms themselves are described in Section 3.4. Next, Section 3.5 details the specific musical features used in our experiments, whose results are presented in Section 3.6. Finally, we conclude in Section 3.8 with a summary and directions for future work.

3.2 Overview

In this section, we give an informal overview of the machine improvisation approach that we developed, sketching the different components using a simplified formalization and a small example. As sketched in Figure 3.1, the overall flow begins with a *training* set of songs $\mathfrak{D} = \{\mathfrak{s}_1, \ldots, \mathfrak{s}_N\}$ and a *reference* song $\mathfrak{s}$, and produces as output a set of specifications used in a *controller*, and an *improviser*, which is the actual object generating new improvisations, i.e., new sequences of notes. The improviser and the procedure to enforce the specifications are implemented closely following [36]: the main contribution of the present paper is how to *generate* the specifications.

All songs are assumed to be in *lead sheet* format, i.e. with a single instrument melody line and an accompaniment specified as a simple chord sequence. In the following, assume $\mathfrak{s}$ is given by:

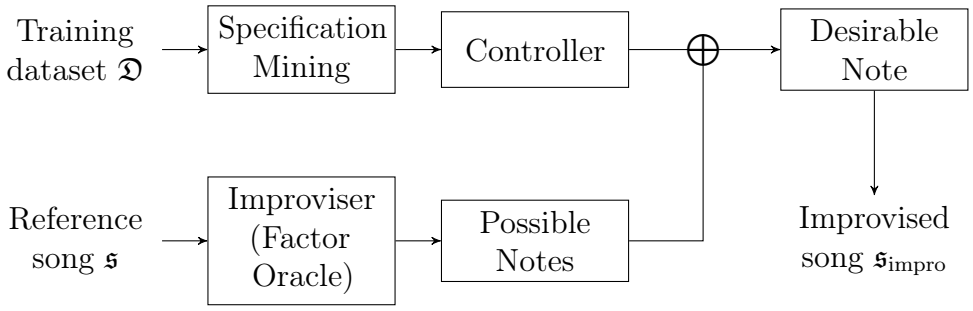
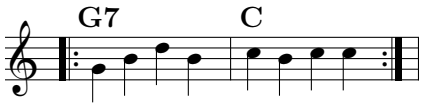


Figure 3.1: Workflow of our approach.

$\mathfrak{s} =$ 

Using standard chord and pitch notations, such as **C**, **Am**, **D7** and a, a#, b, c, c#, d, etc., we can write $\mathfrak{s}$ as a sequence of pitch and chord pairs:

$$(\mathtt{g},\mathbf{G7})(\mathtt{b},\mathbf{G7})(\mathtt{d},\mathbf{G7})(\mathtt{b},\mathbf{G7})(\mathtt{c},\mathbf{C})(\mathtt{b},\mathbf{C})(\mathtt{c},\mathbf{C})(\mathtt{c},\mathbf{C})$$

We say that $\mathfrak{s}$ can be encoded using two *alphabets*, one for chords and another for pitches. Here, for the sake of simplicity, we ignore duration information and other nuances that can be encoded using other alphabets.

3.2.1 Specifications

The songs $\mathfrak{s}_1, \ldots, \mathfrak{s}_N$ in our training set can be represented in the same way and we assume that they all satisfy some *a priori* unknown set of *specifications*, which are properties of the combined sequence of pitches and chords. Simple examples of such specifications include:

- φ_1: the current pitch belongs to the current chord, e.g.,
  - (c,**C**),(g,**C**),(f,**G7**),(b,**G7**) satisfies φ_1
  - (c#,**C**),(a,**C**),(c,**G7**),(e,**G7**) does not satisfy φ_1

- φ_2: the current pitch does not belong to the chord, but the one before did and the one after will, and they are at an interval not greater than one tone, e.g., (c,**C**),(b,**C**) is a sequence satisfying φ_2 *iff* the next note is (c,**C**).

- φ_3: about 70% of the time, a (g,**C**) is followed by a (c,**C**).

In [36], such specifications were described and implemented manually, whereas in this work, we describe how to explicitly and implicitly[1] extract these specifications from the training set $\mathfrak{D}$. Note also that φ_1 and φ_2 are non-probabilistic (hard) specifications and φ_3 is probabilistic (soft). In [36], we only considered non-probabilistic specifications.

The purpose of the improviser is to generate new songs of arbitrary length, e.g.,

(g,**G7**)(b,**G7**)(d,**G7**)(b,**G7**)(c,**C**)(d,**C**)(e,**C**)(c,**C**)(b,**G7**)(a,**G7**)(g,**G7**)(f,**G7**)(e,**C**), ...

that satisfy several criteria, which we state informally below:

(a) All generated sequences satisfy at least one of the non-probabilistic specifications at all times;

(b) The distribution of generated sequences is sufficiently diverse (i.e. there is a variety of different improvisations);

(c) The melody diverges from the reference melody in some controllable way, i.e. it can be made very similar or arbitrarily different;

(d) The distribution of generated sequences satisfies the probabilistic specifications.

In Section 3.3 we will see how criteria (a), (b), and (c) naturally fit into the framework of control improvisation. This is not the case for criterion (d), since [36] did not consider probabilistic specifications.

3.2.2 Factor Oracle-based improvisation

To construct an improviser satisfying the above criteria, we start by following the approach presented in [12]: we construct the *factor oracle* [27] corresponding to the melody line of the reference song $\mathfrak{s}$. A factor oracle is a finite state machine with $n+1$ states (where n is the number of notes) and edges labeled with the pitches of the melody. The factor oracle corresponding to the above reference melody is shown in Figure 3.2. It is constructed in such a way that if one follows the edges and reads labels, it produces a sequence which is a concatenation of subsequences of the reference sequence. Moreover, if one takes only "direct" transitions, i.e. those from a state i to $i+1$, and no other forward or backward transitions, then the sequence of labels reproduces exactly the original sequence. It was then observed in [12,36] that by assigning a fixed probability p, called the *replication probability*, to direct transitions, and uniform probabilities to other "branching" transitions, one obtains a stochastic generator which produces sequences similar in some sense to the original sequence, where the degree of similarity is controlled by p. Such a generator satisfies criteria (b) and (c) above for appropriate values of p.

[1] In our system, the pattern φ_1 is learned implicitly given chord degree specifications.

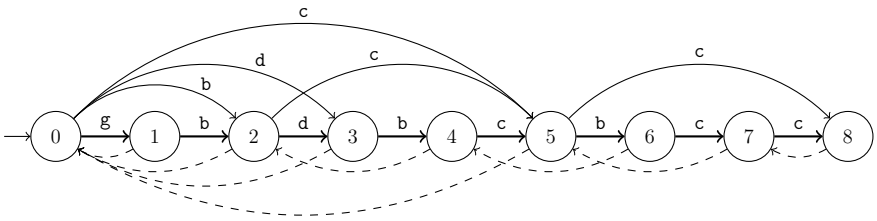


Figure 3.2: Factor oracle constructed from the example melody.

3.2.3 Enforcing Specifications

The factor oracle improviser we just described generates notes without taking into account the harmonic context, and more generally the type of (musical) specifications that we are interested in. Extending our work in [36], and as described in Section 3.4, our approach works by enforcing the desired or mined specifications over sequences of notes proposed by the factor oracle. For example, without specifications the factor oracle of Figure 3.2 might generate the sequence `gbdcb` by going through states 0, 1, 2, 3, 0, 5, and 6 . Once combined with the chord sequence, we get (g,**G7**)(b,**G7**)(d,**G7**)(c,**G7**)(b,C) which clearly violates both φ_1 and φ_2 above, since `c` does not belong to the **G7** chord, and `b` does not belong to the **C** chord. In this situation, our approach would have prevented this improvisation by blocking the transition in the factor oracle from state 5 to 6, forcing the improviser to either take a `c` transition (valid because this would cause the last three notes to satisfy φ_2) or to go back silently to state 0 and take another transition satisfying either φ_1 or φ_2.

3.3 Control Improvisation and Specification Mining

3.3.1 Control Improvisation

We now describe more formally the automata-theoretic concepts used in this research, including the control improvisation problem. For a fully formal definition and theoretical treatment of control improvisation, see [46].

3.3.1.1 Notation and Background

As will be discussed later in Section 3.5, in this research we work entirely with discrete, symbolic representations of musical data (pitches, durations, chords, etc.). Objects such as melodies are represented as finite sequences, or *words*, whose elements are drawn from a finite *alphabet* of symbols Σ. We write ϵ for the empty word consisting of no symbols, and $|w|$ for the length of a word w. Words can be combined

by concatenation, which we denote as multiplication: for example, ab is a word of length 2 if a and b are symbols in Σ.

A convenient formalism for expressing sets of words is *regular expressions*. The simplest regular expressions are written a for some $a \in \Sigma$, and denote the set of words consisting of the single word a. We also use Σ to denote the set of all of these singleton sets. These basic expressions can then be combined using three operators: *concatenation*, *union*, and *Kleene star*. Given regular expressions p and q, their concatenation pq simply consists of all words which are concatenations of a word in p with a word in q. The union $p \cup q$ is just the set-theoretic union, consisting of all words in either p or q. Finally, the Kleene star p^* is the set of all concatenations of finitely many words in p (including the empty concatenation ϵ). For example, Σ^* is precisely the set of all words over the alphabet Σ, and $(a \cup b)^*$ is the set of all words using only the symbols a and b.

Another useful way to represent sets of words is with finite state automata:

Definition 6. *A finite state automaton (FSA) is a tuple* $\mathcal{A} = (Q, q_0, F, \Sigma, \rightarrow)$ *where* Q *is a set of* states*,* $q_0 \in Q$ *is the* initial state*,* $F \subset Q$ *is the set of* accepting states*,* Σ *is a finite set called the* alphabet *and* $\rightarrow \subset Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ *is the* transition relation*. We use the notation* $q \xrightarrow{\sigma} q'$ *to mean that* $(q, \sigma, q') \in \rightarrow$*.*

A word $\sigma_1\sigma_2 \ldots \sigma_n$ is a *trace* of a FSA $\mathcal{A}$ *iff* there exists a sequence of states $q_i \in Q$ such that $q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_{n-1}} q_{n-1} \xrightarrow{\sigma_n} q_n$. It is an *accepting trace* of $\mathcal{A}$ *iff* q_n is in F. The *language* of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of accepting traces of $\mathcal{A}$.

3.3.1.2 Problem Definition

As described above, control improvisation seeks to generate random variations on a *reference word* w_{ref}, all of which must satisfy a given specification and whose similarity to the reference can be controlled. Following [36], the possible variations are given as the language of a plant FSA $\mathcal{A}^p$, and the specification is given by another FSA $\mathcal{A}^s$. Dissimilarity to the reference word is measured by a *divergence measure* $d_{w_{\text{ref}}}$, a nonnegative function on words such that $d_{w_{\text{ref}}}(w_{\text{ref}}) = 0$. These together with parameters indicating how much randomness and similarity to w_{ref} is desired specify a control improvisation problem:

Definition 7. *(Control Improvisation Problem) A control improvisation problem* $\mathcal{P}$ *consists of FSAs* $\mathcal{A}^p$ *and* $\mathcal{A}^s$ *with a common alphabet* Σ*, an accepting trace* w_{ref} *of both* $\mathcal{A}^p$ *and* $\mathcal{A}^s$*, a divergence measure* $d_{w_{ref}}$*, an interval* $I = [\underline{d}, \overline{d}]$*, and parameters* $\varepsilon, \rho \in (0, 1)$*. A solution of* $\mathcal{P}$ *is a probabilistic algorithm generating words* w *in* Σ^* *such that the following conditions hold:*

(a) Safety: *each* w *is an accepting trace of both* $\mathcal{A}^p$ *and* $\mathcal{A}^s$*;*

(b) Randomness: *the probability of generating each* w *is smaller than* ρ*;*

(c) Bounded Divergence: $\Pr(d_{w_{ref}}(w) \in [\underline{d}, \overline{d}]) > 1 - \varepsilon$.

To illustrate how this problem is useful, let us cast the example of Section 3.2 as an instance of control improvisation. Recall that songs were represented as sequences of pitch and chord pairs: thus our alphabet Σ consists of all possible such pairs. The plant automaton $\mathcal{A}^p$ encodes a model for generating improvisations without enforcing any specifications, which in our case is a factor oracle over the reference song (as described in Section 3.7). The automaton $\mathcal{A}^s$ depends on which specifications we desire. For the specification φ_1 from Section 3.2, for example, we might use the automaton in Figure 3.3. To simplify the diagram we have consolidated some transitions: from q_0 there are separate transitions for input symbols $(\mathsf{c}, \mathbf{C})$ and $(\mathsf{g}, \mathbf{C})$, for example, and generally for all pairs where the pitch is contained in the chord, but since all these transitions lead back to q_0 we have drawn them as a single arrow.

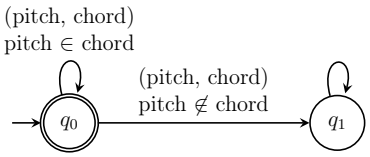


Figure 3.3: Specification automaton for φ_1.

Now we can see how requirements (a), (b), and (c) on a generated word in the control improvisation problem correspond exactly to the requirements stated in Section 3.2 for improvised songs:

(a) The improvisation being an accepting trace of $\mathcal{A}^p$ ensures it can be generated by the underlying specification-free improvisation system (e.g. a factor oracle), while being an accepting trace of $\mathcal{A}^s$ ensures that it satisfies our specifications[2].

(b) By requiring that each improvisation w be generated with probability at most ρ, we ensure that at least $1/\rho$ improvisations can be generated. So by making ρ small we can ensure a diverse distribution of improvisations.

(c) By defining the interval I and parameter ε appropriately, this condition allows us to control how much the improvisations can diverge from the reference song (according to the similarity metric used).

This leaves only requirement (d) from Section 3.2, namely enforcement of probabilistic specifications. This does not fit into the definition of control improvisation as stated above, and so will be discussed in Section 3.4.2 below.

[2]In Section 3.2 we required that the improvisation satisfy *at least one* of several specifications at each event, but those can easily be combined into a single specification that is required to hold over the entire improvisation.

3.3.2 Specification

In this current work, we expand our previous efforts in [36] by developing an inference engine that mines specifications from a song dataset in the form of *pattern graphs* learned using a set of pre-defined pattern templates. The following paragraphs adapt the work of [82] to formally describe specification mining in music.

3.3.2.1 Events and Patterns

Let F be the set of feature vectors extracted from a song $\mathcal{S}$, e.g. pitch, duration, and so forth. For every feature $f \in F$, we use the notation $v_{f,t}$ to indicate the valuation of f at time t.

Definition 8 (Event)**.** *An event is a tuple $(\vec{f}, \vec{v}, t)$, where $\vec{f}$ is a set of musical features and $\vec{v}$ is their corresponding valuations at time t. The alphabet Σ_f is the set of possible events for feature f, and a finite trace τ is a sequence of events ordered by their time of occurrence. We address monophonic music in this research, so there is only one event at a time.*

Definition 9 (Projection)**.** *The projection $\pi_\Sigma(\tau)$ of a trace τ onto an alphabet Σ is defined as τ with all events not in Σ deleted.*

Definition 10 (Specification Pattern)**.** *A specification pattern is an FSA over symbols Σ. Patterns can be parametrized by the events used in this alphabet; for example, we use "the $\mathcal{A}$ pattern between events a and b" to indicate the pattern obtained by taking an FSA $\mathcal{A}$ with $|\Sigma| = 2$ and using a as the first element of Σ and b as the second. A pattern* occurs[3] *in a trace τ with alphabet $\Sigma_\tau \supseteq \Sigma$ if and only if there is a subword σ of τ such that $\pi_\Sigma(\sigma) \in \mathcal{L}(\mathcal{A})$.*

Definition 11 (Binary Pattern)**.** *A specification pattern with alphabet size 2. We denote a binary pattern between events a and b as a $\boldsymbol{R}$ b, where $\boldsymbol{R}$ is a label identifying the pattern.*

Now that we have described patterns as a general concept, we define three types of patterns that we will use in this research. Each pattern corresponds to common musical behaviors such as harmonic resolutions and ornaments. For simplicity we define them using regular expressions, which are equivalent to FSAs.

Followed (F): The followed pattern between two events a and b occurs when a is immediately followed by b. It provides information about possible transitions between events, which can be used, for example, to specify the resolution of non-chord tones. We

[3] Note that this is different from the trace *satisfying* the pattern in the sense of [82]: we are interested in occurrences of patterns *within* a trace, whereas they require the entire trace to match the pattern.

denote the followed pattern as a **F** b and can match it with the regular expression (ab).

'Til (T): The 'Til pattern between two events a and b occurs when a occurs two or more times in sequence and is then immediately followed by b. Compared to the followed pattern, it provides more specific information about what transitions are possible after self-transitions are taken. We denote this pattern as a **T** b and can match it with the regular expression (aaa^*b).

Surrounding (S): The surrounding pattern between two events a and b occurs when event a immediately precedes and succeeds event b. It provides information over a time-window of three events and we musically describe it as an ornamented self-transition. We use a **S** b to denote this pattern and can match it with the regular expression (aba).

3.3.2.2 Pattern Merging

If every match to a pattern $P_2 = a$ **R** b occurs inside a match to a pattern $P_1 = a$ **Q** b, we say that P_1 *subsumes* P_2 and write $P_1 \implies P_2$. When this happens, we only add the *stronger* pattern P_1 to the pattern graph. The purpose of merging is to emphasize longer musical structures: if one pattern always occurs only as part of a longer one, then we will only allow the longer pattern to occur in our generated phrases, but not the shorter pattern by itself.

An example is the chord degree[4] specification mined from the song *Crossroads Blues*, shown in Figure 3.4. Here, chord degree 10 (note f) is followed by chord degree 7 (note d), so without merging we would learn the pattern *10* **F** *7*. This would allow generating words such as $(10, 7, 10, 7)$, which is inconsistent with the melodic motives in the song, which always have *multiple* occurrences of 10 before transitioning to 7. Thus every match to *10* **F** *7* is contained in a match to *10* **T** *7*, and so with pattern merging we only learn *10* **T** *7*, thereby forbidding $(10, 7, 10, 7)$. In fact, *10* **T** *7* shows how a pattern can subsume multiple patterns, since in this example it also subsumes *10* **F** *10* and *10* **T** *10* (both of which we would otherwise learn).

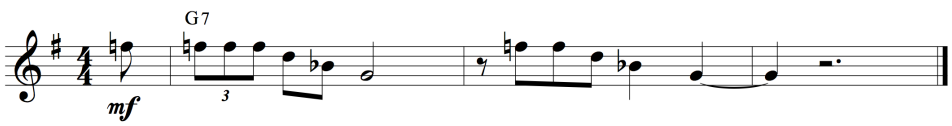


Figure 3.4: First phrase of Crossroads Blues by Robert Johnson as transcribed in the Real Book of Blues.

[4]In this research, chord degree is represented by the distance, in semitones, from a note to the current chord's root note.

3.3.2.3 Specifications from Patterns

For each feature $f \in F$, the specifications on f that we mine are of several different types:

1. Which values of f can occur at the beginning of a phrase.
2. Which values of f can occur at the end of a phrase.
3. Which patterns over Σ_f can occur in the phrase.
4. The empirical probabilities of these patterns.

More formally, the type (3) specification requires that every pattern that matches the trace either is allowed (i.e. occurred in the training data) or is subsumed by one that is allowed. For example, suppose the specification was learned from the single word (a, b, a, b, a). Due to merging, we only learn the pattern a **S** b, even though for example the word matches b **F** a. Now consider the word (b, a). The only match to any pattern in this word is the entire word itself, which matches b **F** a. This pattern was not learned, and is trivially not subsumed by a learned pattern since there are no other matches to subsume it. Therefore (b, a) does not satisfy the specification. However, the word (a, b, a) does satisfy the specification: it matches a **S** b, which was learned; it also matches a **F** b and b **F** a, but both matches are subsumed by the match to a **S** b.

The first three types of specification are hard constraints that the phrases we generate must respect, while the last can be viewed as a kind of soft constraint. We encapsulate all four types in a data structure we call the *pattern graph*.

Definition 12 (Pattern Graph)**.** *A pattern graph is a labelled directed multigraph whose nodes are elements of Σ_f, i.e. values of a feature f. A node can be labelled as a starting node, an ending node, or neither. Edges are labelled with a type of binary pattern and a count indicating how many times the pattern occurred in the dataset.*

For example, an edge (a, b) labelled $(\mathbf{R}, 3)$ in the pattern graph means the pattern a **R** b occurred 3 times in the dataset. A complete example of a pattern graph is shown in Figure 3.5, where we have indicated starting nodes with an unlabelled incoming arrow and ending nodes with a double circle (by analogy to the standard notation for FSAs).

While a pattern graph represents the hard specifications above, it is not itself an automaton. However, our method still fits into the framework of control improvisation as presented in Section 3.3.1.2, because the pattern graph can be converted into a specification automaton $\mathcal{A}^s$. In fact, as explained in Section 3.4.2, our improvisation algorithm does not need to perform this conversion.

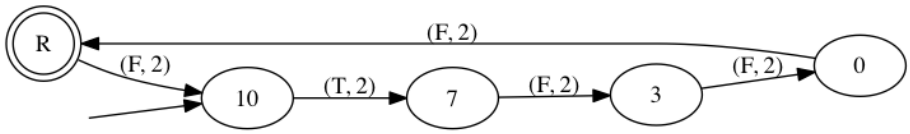


Figure 3.5: Pattern graph learned on the chord degree feature (interval from root) extracted from the phrase in Fig. 3.4.

3.4 Learning and Enforcing Specifications

In this section we describe how we learn hard and soft specifications in the form of pattern graphs, and how those graphs are used to guide the improvisation process.

3.4.1 Learning Specifications

In this subsection we describe the procedure used to learn pattern graphs from a dataset $\mathfrak{D}$ with features $\mathcal{F}$. The procedure also takes as input a set $\mathcal{P}$ of patterns, consisting of a function for each pattern type $\mathbf{R}$ that maps feature values a, b to a regular expression defining the pattern a $\mathbf{R}$ b. As a preprocessing step, the songs in $\mathfrak{D}$ are segmented into three phrases (A A' B)[5]. An example of a segmented dataset is shown in Table 3.1.

After segmentation, for each feature f a pattern graph $\mathcal{G}_f$ is constructed by Algorithm 1, whose main steps are as follows. First, for each phrase we add the first and last feature values as starting/ending nodes respectively in the graph. Second, for every $a, b \in \Sigma_f$ we find all matches to a $\mathbf{R}$ b for every pattern type $\mathbf{R}$. If there is a match to a $\mathbf{R}$ b which is not subsumed, then we add a corresponding edge to the pattern graph labelled with the number of times the pattern occurs. A naïve implementation of this algorithm could explicitly compute the locations of every match by instantiating all possible regular expressions for the patterns in $\mathcal{P}$ and finding all ways each expression matches the phrase. Then the subsumption check is simply a matter of comparing the locations of the matches. If the number of feature values or pattern types is large, this could be highly inefficient, but for the pattern types used in this research all non-subsumed patterns can be found with a fast linear search.

3.4.2 Improvising with Specifications

The core of our improvisation approach is the factor oracle (FO), briefly described in Section 3.2. Following [36], we enforce specifications on top of the factor oracle by solving a control improvisation problem where the plant $\mathcal{A}^p$ encodes the factor

[5]Phrase boundaries are automatically extracted and manually corrected if necessary.

Algorithm 1: Specification Mining Algorithm

Input: dataset $\mathfrak{D}$ over features $\mathcal{F}$; patterns $\mathcal{P}$
Output: a pattern graph $\mathcal{G}_f$ for each $f \in \mathcal{F}$

```
for f ∈ F do
    G_f ← new pattern graph on vertices Σ_f
    for song ∈ 𝔇 do
        for phrase ∈ song do
            phrase_f ← the sequence of values of the feature f in phrase
            label the first element of phrase_f as a starting node in G_f
            label the last element of phrase_f as an ending node in G_f
            for a, b ∈ Σ_f do
                counts ← countPatternMatches(a, b, phrase_f, P)
                foreach pattern P with counts(P) > 0 do
                    add to G_f the edge (a, b) with label (P, counts(P))
                end
            end
        end
    end
end
```

oracle built from w_{ref} and enforces its chord progression. As described in [36], if the specification automaton $\mathcal{A}^s$ is non-blocking in the sense that an accepting state is always reachable, i.e. there are no deadlocks, we can solve the control improvisation problem by restricting the factor oracle to only take transitions consistent with $\mathcal{A}^s$. If there are no such transitions in the factor oracle, we use heuristics to decide which destination state is most appropriate. This procedure can be extended to general specifications using techniques from supervisory control (see [36]).

As mentioned above, the pattern graphs learned by the algorithm in the previous section can be converted into a specification automaton $\mathcal{A}^s$. However, this construction involves taking the product of many automata — one for each pattern — resulting in a final automaton whose size grows exponentially with the number of patterns. So in practice $\mathcal{A}^s$ is likely to be too large to construct explicitly. Therefore, we use the following heuristic: we assume $\mathcal{A}^s$ is non-blocking, and if we reach a blocked state (i.e. one with no outgoing transitions), we reject our current improvisation and start over. Then we can use the procedure for non-blocking automata cited above, which only requires being able to compute for any state of $\mathcal{A}^s$ the set of outgoing transitions. As we will show below, this information can be read off from the pattern graph without having to construct $\mathcal{A}^s$. In practice we find that reaching a blocked state is rare, so few restarts are required and this procedure is efficient.

Determining the transitions allowed by $\mathcal{A}^s$ from the current state is straightforward. At the beginning of an improvisation, we only allow symbols which are labelled as starting nodes in the pattern graph. Likewise, we only allow an improvisation to end on symbols labelled as ending nodes. Finally, if the last generated symbol was a, a transition on symbol b is allowed only in the following situations:

1. the pattern graph has an edge from a to b labelled with pattern $\mathbf{F}$;
2. the pattern graph has an edge from a to b labelled with pattern $\mathbf{S}$; in this case we require the next transition to be on symbol a;
3. the pattern graph has an edge from a to b labelled with pattern $\mathbf{T}$, and the symbol before a was also an a (as the a $\mathbf{T}$ b pattern requires two or more copies of a);
4. $a = b$, i.e. the transition would generate another a, and the pattern graph has an edge from a to any symbol c labelled with pattern $\mathbf{T}$ (since a $\mathbf{T}$ c allows arbitrarily many copies of a prior to c) .

It is easy to see that this method correctly enforces our specification $\mathcal{A}^s$: the generated words match only patterns that occur in the pattern graph or are subsumed by such patterns.

As an example, consider the pattern graph built from the single word *aaab*. Because of pattern merging, the graph will only have a single edge, from a to b and labelled $(\mathbf{T}, 1)$. Initially, we only allow a transition on a since it is the only node labelled as a starting node. Next, situations (1), (2), and (3) above do not hold (the last because we have not generated any symbol prior to the a), but situation (4) does and allows another transition on a. Now by (3) and (4) we can transition on either b or a respectively — suppose we choose the former (as we will discuss below, we actually pick between transitions randomly if more than one is available). Since b is labelled as an ending node in the graph, we can stop here with the improvisation *aab*.

One remaining question is how to pick the transition to follow in the factor oracle when more than one choice is consistent with $\mathcal{A}^s$. This is where we incorporate the probabilistic or "soft" specifications mentioned in Section 3.2. Building on a suggestion in [36], we randomly sample from the consistent transitions: a direct transition gets the replication probability p, and the other transitions get probabilities related to their empirical probabilities in the dataset. Specifically, the probability for a non-direct transition is computed as follows: we sum the counts in the pattern graph for every edge that can allow the transition according to the rules above, and assign a probability proportional to this sum.

To illustrate this computation, consider the pattern graph learned from the word *aabcaabcaa*, and say we have generated *aa* so far. The pattern graph has an edge from a to b labelled $(\mathbf{T}, 2)$, and an edge from a to a labelled $(\mathbf{F}, 1)$. According to the rules

above, the first edge allows us to transition on b (by (3)), and both edges allow us to transition on a (by (4) and (3) respectively). Adding up the corresponding counts, we assign probabilities to b and a proportional to 2 and $2 + 1 = 3$ respectively (assuming neither transition is the direct transition in the factor oracle). Normalizing, we will pick b with probability $2/5$ and a with probability $3/5$.

The goal of this heuristic is produce improvisations whose feature distribution is more similar to that of the dataset than would be achieved with a purely random choice of transitions (see Section 3.6 for qualitative experiments assessing this). Many other heuristics are possible, and could give better results in some circumstances. For example, under the simple heuristic above the pattern a **T** b contributes equally to the probabilities of transitions on a and b, thereby prioritizing short 'Till patterns. A more sophisticated heuristic could incorporate the number of repetitions of a inside each instance of the pattern, adjusting the transition probabilities accordingly. We also note that random transition heuristics of this kind can be thought of as attempts to enforce a type of divergence criterion similar to the one used in the control improvisation problem, but where divergence is measured against a dataset of multiple songs instead of a single reference word.

In summary, the overall process for generating an improvisation of length at least n is:

1. Maintain a sequence $(q_0, s_0)(q_1, s_0) \dots (q_k, s_k)$ of pairs of states of $\mathcal{A}^p$ and $\mathcal{A}^s$, and a word $w_k = \sigma_0\sigma_1 \dots \sigma_k \in \Sigma^k$.
2. If $k \geq n$ and s_k is accepting, return w_k.
3. If there are no outgoing transitions from s_k, restart the improvisation process.
4. Let C be the set of transitions from q_k that are compatible with s_k (i.e. such that s_k has a transition on the same input symbol).
5. If C is empty, we need to add a transition to $\mathcal{A}^p$. Pick a random transition from q_k using the distribution described above; let σ be the input symbol triggering it and set q_{k+1} to be its destination state. Among all input symbols for which there is a transition from s_k, find the one, say τ, most similar to σ (e.g. among pitches, the nearest in terms of intervals), and set s_{k+1} to be the corresponding destination state. Add a transition from q_k to q_{k+1} on input τ, and set $\sigma_{k+1} = \tau$.
6. Otherwise C is nonempty. Pick a random transition from C using the distribution described above; set σ_{k+1} to the input symbol triggering it and q_{k+1} to its destination state. Set s_{k+1} to the destination state of the corresponding transition from s_k.
7. Repeat from step 1.

3.5 Music Specification Mining

In this section we describe some features that can be used to describe expected musical behaviors or properties and, therefore, are appropriate to mine specifications from. We start by formally defining the components involved.

We abstract and formalize a song into a sequence of *melodies*, where a *melody* is defined as a string of pitched notes and rests, aligned with an *accompaniment*, a sequence of chords with given durations[6]. The time unit is the *beat*, including respective integer subdivisions, and the piece is divided into measures, which are sequences of k beats. We assume that the accompaniment is fixed and our goal is to define an improviser for the melody. Hence, the plant will model the behavior of the accompaniment, without constraining the melody, and the specification FSA will set constraints on acceptable melodies played together with the accompaniment. To encode all events in a score, we use an alphabet which is the product of four alphabets: $\Sigma = \Sigma_p \times \Sigma_d \times \Sigma_c \times \Sigma_b$, where

- Σ_p is the *pitches* alphabet, i.e. $\Sigma_p = \{$𝄽, `a0`, `a#0`, `b0`, `c0`, $\cdots\}$;
- Σ_d is the *durations* alphabet, i.e. $\Sigma_d = \{$𝅘𝅥𝅯, ♩, 𝅗𝅥, $\ldots\}$ with ♩ $= 1$ beat. Note that Σ_d also includes fractional durations, e.g., for triplets, as discussed below;
- Σ_c is the *chords* alphabet, i.e. $\Sigma_c = \{$C, C7, G, Emaj, Adim, $\ldots\}$;
- Σ_b is the *beat* alphabet. For example, if the smallest duration (excluding fractional durations) is the eighth note, i.e. half a beat, then $\Sigma_b = \{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5\}$, where 0 represents the beginning of the first beat in the measure.

Note that the full alphabet enables the creation of data abstractions, such as melodic intervals and tone classes. A similar strategy is used in [28], where data abstractions (derived types) specific for chorales are implemented. In our current implementation, all pattern graphs implicitly use the full alphabet Σ. However, each component alphabet is meant to address one particular aspect of the music formalization, and we construct the specifications by composing many small specifications which operate only on some of the component alphabets. For example, a specification might constrain only the sequence of beats in the melody, without using the other information in each event, and so could be represented as a small pattern graph defined only over Σ_b.

3.5.1 Time Domain Features

- **Event Duration**: This feature describes the duration, given in beats, of silences and tones. The event duration feature imposes hard constraints on duration

[6] This is not canonical, and dynamics are not considered in this work, although they could easily be treated as another feature.

diversity but provides only weak guarantees on rhythmic complexity because it has no awareness of beat location. Figure 3.6 provides one example where specifications built on this feature fail to prevent incomplete tuplets. We can impose further constraints on rhythmic complexity by combining the features event duration and beat onset location.

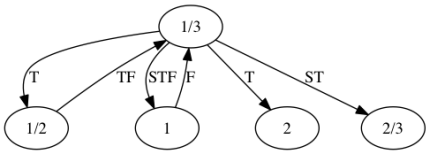


Figure 3.6: Selection of event duration specifications learned from the training set. The pattern 1/3 **S** 1 (1/3, 1, 1/3) is allowed but can produce incomplete tuplets if placed on certain beats.

- **Beat onset location**: This feature describes where events happen within the beat, ignoring information about the length of the event. It is computed by taking the remainder modulo 1 of the beat feature, which describes the onset locations of each event. Cooperatively, event duration and beat onset location specifications impose hard constraints on rhythmic complexity that duration specifications alone do not guarantee, and allow for rhythmic diversity that beat onset location alone does not guarantee. These specifications extend our work in [36] by replacing handmade specifications designed to ensure rhythmic tuplet completeness with mined specifications. Figure 3.7 shows an example of the patterns learned and respective patterns between beat onset locations.

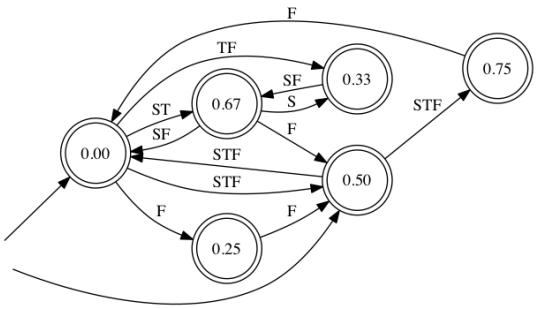


Figure 3.7: Beat onset locations specifications learned from the training set.

3.5.2 Frequency Domain Features

- **Scale Degree**: The scale degree is the identification of a note disregarding its octave but regarding its distance from a reference tonality. We represent scale

degree numerically, e.g. in a C scale $C = 0, C\# = 1, \ldots B = 11$. Songs usually impose soft constraints on the pitch space, defining the set of appropriate scale degrees and transitions thereof. The selection of specifications mined from scale degree shown in Figure 3.8 conform with the general consent that blues songs include the main key's major scale with the "flat seven" (scale degree 10) and the blue note (scale degree 3), excluding, for example flat ninths (scale degree 1) so common in jazz literature. In Figure 3.8, notice that sharp fourths (scale degree 6) are used as approach tones to scale degree 5 and 6. Since scale degree can only provide overall harmonic constraints to each tone over the scope of the entire song, we use another feature to provide harmonic constraints based on chord progression, therefore increasing the temporal granularity of the harmonic specifications.

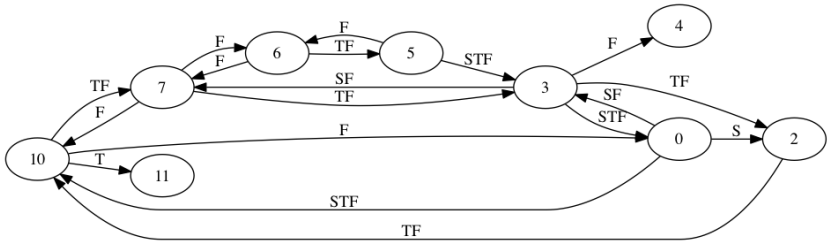


Figure 3.8: Selection of scale degree specifications learned from the training set.

- **Interval Classification**: Expanding on [36], we replace the hand-designed tone classification specifications, here called interval classification, with mined specifications. Our specifications include information about the size (diatonic or leap) and quality (consonant or dissonant) of the music interval that precedes each tone. Figure 3.9 illustrates the mined specifications. We use the symbols A, B, C, and D, to describe tones reached by consonant step, consonant leap, dissonant (non-chord tones) step, and dissonant leap respectively. Consonant and dissonant notes preceded by rests are described with the symbols I and O respectively. The symbol R represents rests. Although scale degree and interval classification specifications ensure desirable harmonic guarantees given key and chord context, they provide no guarantees over the contour of a melody.

- **Chord Degree**: The chord degree is the identification of a note regarding its distance in semitones to the root of a chord. It adds harmonic specificity to the interval class, without enforcing a melodic contour.

- **Melodic Interval**: This feature operates on the first difference of pitch values and is associated with the contour of a melody. Combined with scale degree and interval classification, it provides harmonic and melodic constraints, including melodic contour.

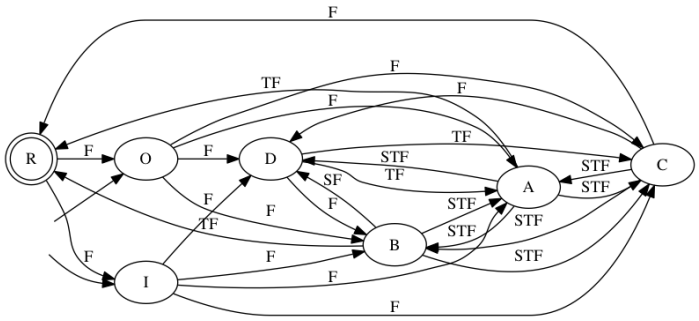


Figure 3.9: Interval class specifications learned from the training set.

	chord	dur	measure	phrase	...	pitch	mel_interval	beat	interval_class
0	F7	14/3	1	1	...	69	NA	1	I
1	F7	1/3	2	1	...	65	-4	5/3	B
2	F7	2/3	2	1	...	67	2	1	C
3	F7	1/3	2	1	...	65	-2	5/3	A
4	F7	1	2	1	...	68	3	1	D
...	...	...	...	...	...	...	...	.....	
22	B-7	1	10	3	...	68	3	1	B
23	B-7	1	10	3	...	67	-1	1	C
24	F7	4	11	3	...	65	-2	1	A
25	F7	-4	12	3	...	NA	NA	1	R

Table 3.1: Dataframe from Blues Stay Away From Me by Wayne Raney et al. NA represents a rest or a transition to or from a rest. *dur* is duration in beats and *mel_interval* is melodic interval

Table 3.1 provides the reader with a selection of features extracted from a blues song, including chord and phrase number annotations. The next section analyzes in detail the application of specification mining to the tasks of song validation and machine improvisation with formal specifications.

3.6 Experimental Results

In all our experiments, we used the features above to mine specifications from a training set of 20 blues songs, $\mathfrak{D}^{train}$, digitized from the Real Book of Blues [84].

3.6.1 Specification Validation

To check whether our learned specifications are overfitting the training set, we measured the extent to which a disjoint set of songs from the same genre satisfied the specifications. We divide specification violations into two categories:

- the word has a pattern whose symbols exist in the alphabet but the pattern is not allowed by the specification.
- the starting/ending node does not exist in the specification.

An example of the first type is playing an interval that is not valid, although both notes exist in the scale; e.g. the augmented fourth is a forbidden interval in harmony or has to be appropriately resolved. The second type ensures that phrases will start with the proper notes.

We quantify how a test song violates the specification by computing the *violation ratio*, which is the fraction of patterns occurring in the song that are illegal. This quantity is computed by building the pattern graphs for the test song and comparing them with the corresponding graphs learned from the training set. Although the violation ratio is a quantitative measure of how badly the specifications are violated, its formulation is not based on human cognition and perception. Ideally, we would like a measure that takes into account how the violations perceptually differ from the behaviors in the specification. We have developed a prototype of such measure, but leave its evaluation to future work.

In our experiments, we used a separate test set $\mathfrak{D}^{test}$ consisting of 10 blues songs, digitized from the Country Blues songbook [58]. In total there were 975 patterns learned from $\mathfrak{D}^{test}$, of which 124 were violations. In particular, there were 51 chord degree violations, 47 melodic interval violations and 26 interval class violations, yielding a violation ratio of 0.12 for the $\mathfrak{D}^{test}$ dataset with 10 songs. All songs in $\mathfrak{D}^{test}$ had starting and ending nodes that existed in the specification.

Figure 3.10 provides histograms of violations obtained by using harmonic specifications based on chord degree and interval to validate each song in the test set.

Given the small size of our training data for learning specifications, we assume that these violations would not occur on a larger training set. This validation can be exploited in style recognition and we foresee that more complex validations are possible by creating more elaborated metrics and using a combination of specifications from multiple styles.

Overall, there were many interval violations related to leaps. Although both training and test sets had licks that used chord arpeggiations, their starting notes were different, leading to invalid interval transitions. We provide a specific example in Figure 3.10 where the first three notes represent an arpeggiation over E7 that starts with an invalid interval. In addition to interval violations, this test set has chord degree violations that are mainly related to playing in sequence two notes that do not

belong to the current chord. After analysis, we learned that these invalid transitions occur in blues songs where the second phrase is a repetition of the first phrase under a different harmony. The blues song *You Don't Mean No Good* in Figure 3.10 has a good example. In that song, the E7 arpeggio on the first phrase, measure 2, is valid under E7 but invalid on the second phrase, measure 6, under A7.

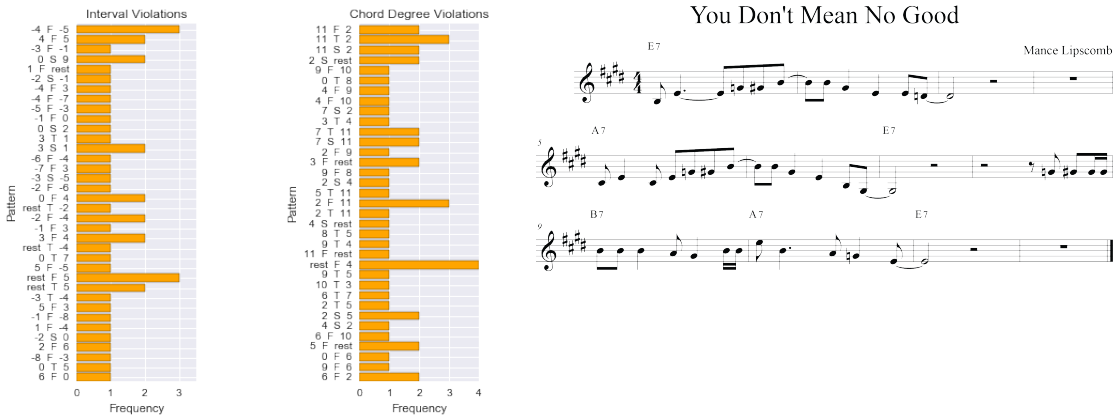


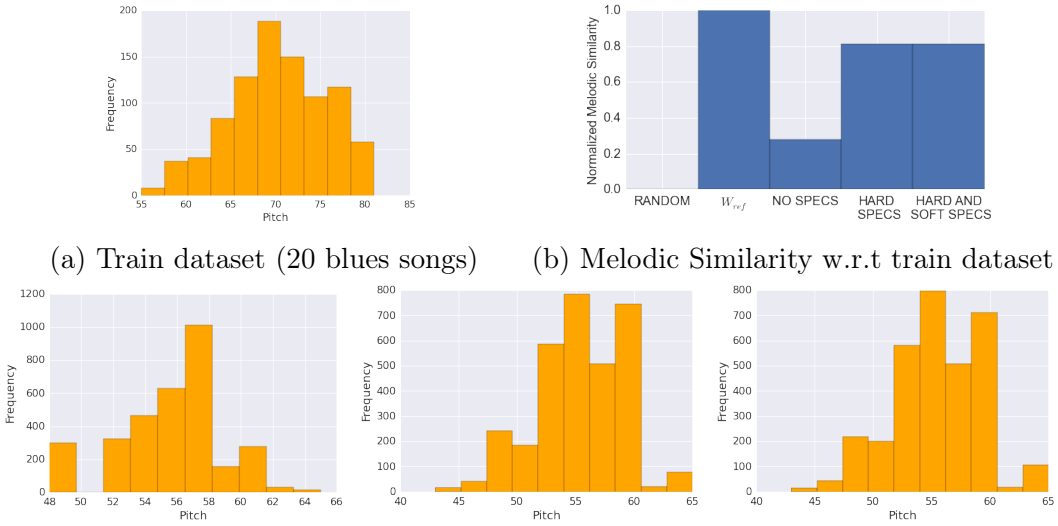
Figure 3.10: Histogram of melodic interval and chord degree violations. The y-axis represents the patterns that do not exist in the specification and the x-axis represents their frequency. F and T represent the patterns Followed and 'Till respectively.

3.6.2 Machine Improvisation with hard and soft specifications

Using the 12-bar blues excerpt and its chord progression shown in Figure 3.12, we generated improvisations with and without specifications, generated from $\mathfrak{D}^{train}$, using the factor oracle with 75% replication probability. For this task, we used joint specifications, including duration, beat onset location, chord degree, interval class and melodic interval.

For the quantitative analysis, we computed the average melodic similarity between $\mathfrak{D}^{train}$ and other sets of improvisation, including: 50 factor oracle improvisations generated without specifications, 50 factor oracle improvisations generated with hard specifications and 50 factor oracle improvisations generated with soft and hard specifications. The melodic similarity is computed using the algorithm described in [129]. As baselines, we also computed the similarity of $\mathfrak{D}^{train}$ to the 12 Bar Blues reference word and to 50 songs with random notes and durations.

The results in Figure 3.11 show that the specifications are successful in making the improvisation generated by the factor oracle more similar to the melodies from which the specifications were mined. In the case of the 12-bar blues, the pitch distribution and the melodic similarity of the improvisations generated with hard specs and hard and soft specs to the training data is almost equal.



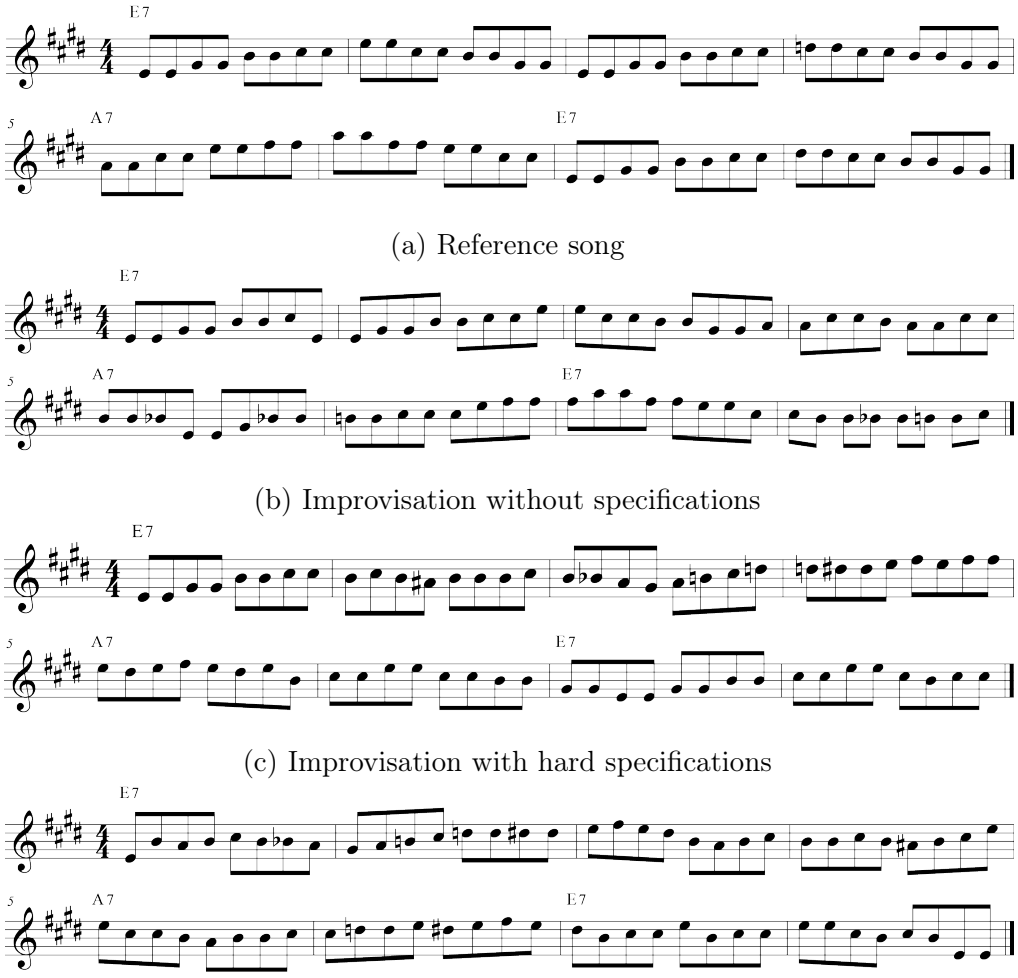
(a) Train dataset (20 blues songs) (b) Melodic Similarity w.r.t train dataset

(c) 50 improvisations without specifications (d) 50 improvisations with hard specifications (e) 50 improvisations with hard and soft specifications

Figure 3.11: Pitch Histograms for the training set and factor oracle improvisations with 0.75 replication probability. The melodic similarity with respect to the training set bargraph (b) has values normalized by the larger similarity value (W_{ref}).

Qualitatively, the improvisation without specifications violates several specifications related to expected harmonic and melodic behavior, as Figure 3.12 confirms. For example, measure 4 in the improvisation without specifications has chord degrees that violate harmonic specifications. This is expected because the transitions taken by the unsupervised improvisation disregard harmonic context, thus commonly producing unprepared and uncommon dissonant notes. In addition, the melodic profile of the unsupervised improvisation is rather jumpy.

Both supervised improvisations are able to keep overall harmonic coherence despite the use of chromaticism. Their melodic contour is rather smooth and the improvisations include several occurrences of the 'Til and Surrounding patterns, as measures 5 and 1 of the improvisation with hard and with both hard and soft specifications respectively show. We noticed that the improvisations generated with the specifications are considerably similar, which implies that there are not many solutions to the constraints enforced by the specifications. This raises an interesting research question, namely how to ensure that there is enough diversity among the improvisations while still satisfying the constraints.



(a) Reference song

(b) Improvisation without specifications

(c) Improvisation with hard specifications

(d) Improvisation with hard and soft specifications

Figure 3.12: Factor Oracle improvisations with 0.75 replication probability on a traditional instrumental blues lick

3.7 Related Work

The concept of *control improvisation* was first introduced in [35,36] and formalized in [45,46]. It was presented as a variation of the standard supervisory control problem for discrete event systems [22], with applications to the generation of music. A real time implementation was presented in [34], and the problem was investigated theoretically in [46]. As indicated in the previous section, the core of the improvisation process rests on the notion of a factor oracle. The factor oracle (FO) was initially introduced in [27] as an algorithm for optimal string matching, and later suggested as a suitable data structure for machine improvisation in [12]. It is in use in several prominent

improvisation systems such as OMax[7] and its variant ImproTek [99].

Specification mining is a form of inductive synthesis in which formal artifacts are synthesized from examples [67,118]. In the software engineering literature, specification mining is an efficient procedure to automatically infer, from empirical data, general rules that describe the interactions of a program with an application programming interface (API) or abstract datatype (ADT) [3]. It has convenient properties that facilitate and optimize the process of developing formal specifications. First, specification mining is either entirely automatic, or only requires the relatively simple task of creating templates. In addition, it can exploit latent properties that are unknown to the user and only reflected in the data, offering valuable information on commonalities in large datasets.

Techniques to automatically generate specifications date back to the early seventies, including [19,132]. More recent efforts that analyze the problem of specification inference include [2,3,43,82]. In general, specification mining tools infer temporal properties in the form of mathematical logic or automata. Broadly speaking, the two main strategies for building these automata include: learning a single automaton and inferring specifications from it; learning small templates and designing a complex automaton from them. For example, [3] learns a single probabilistic finite state automaton from the trace and then extracts likely properties from it. The other strategy circumvents the NP-hard challenge of directly learning a single finite state automaton [52,53] by first learning small specifications and then post-processing them to build more complex state machines. The idea of mining simple alternating patterns was introduced by [43], and several subsequent efforts [47,48,133,138] built upon this work. We forward the reader to [81] for a more in-depth discussion of related work.

In music, and specifically music improvisation, the task of learning or describing such general rules is difficult, even for experts, due to music's large parameter space and richness of interpretation. Therefore, specification mining is very attractive because it offers a systematic and automatic mechanism for learning these specifications from large amounts of data.

3.8 Discussion

We proposed a solution to the problem of mining specifications from symbolic music for machine improvisation with formal specifications. This solution replaced our previous approach, which required manually inferring and encoding specifications, with an engine that automatically mines information from a dataset of songs. Our experiments show that the new approach is successful both in graphically and algorithmically describing characteristics of a music collection, and in guiding improvisations in the style of that music collection.

[7] http://repmus.ircam.fr/omax/home

This research is a first step towards music specification mining and we plan to investigate mechanisms to build more complex specifications, e.g. hierarchical specification schemes where the specification layers have hierarchical relationships such as section, phrase, motif, note. We are also interested in designing an interface that allows a musician to write, mine and combine specifications from MIDI data and visualize pattern graphs extracted from it.

Chapter 4

Attacking Speaker Recognition Systems with Generative Adversarial Networks

4.1 Introduction and Motivation

Speaker authentication systems are being deployed for security critical applications in industries like banking, forensics, and home automation. Like other domains, such industries have benefited from recent advancements in deep learning that lead to improved accuracy and trainability of the speech authentication systems. Despite the improvement in the efficiency of these systems, evidence shows that they can be susceptible to adversarial attacks[136], thus motivating a current focus on understanding adversarial attacks ([127], [56]), finding countermeasures to detect and deflect them and designing systems that are provably correct with respect to mathematically-specified requirements [120].

Parallel to advancements in speech authentication, neural speech *generation* (the process of using deep neural networks to generate speech) has also seen huge progress in recent years ([131], [5]). The combination of these advancements begs a natural question that has, to the best of our knowledge, not yet been answered:

Are speech authentication systems robust
to adversarial attacks by speech generative models?

Generative Adversarial Networks (GANs) are generative models that recently have been used to produce incredibly authentic samples in a variety of fields. The core idea of GANs, a minimax game played between a generator network and a discriminator network, extends naturally to the field of speaker authentication and spoofing. We show that a variant of GAN training motivates the model's use as an attacking architecture.

With regards to this question, we offer in this research the following contributions:

- We evaluate SampleRNN and WaveNet in their ability to fool text-independent speaker recognizers.
- We propose strategies for untargeted attacks using Generative Adversarial Networks.
- We propose strategies for targeted attacks using a new objective function based on the improved Wasserstein GAN.

4.2 Related Work

Modern generative models are sophisticated enough to produce fake[1] speech samples that can be indistinguishable from real human speech. In this section, we provide a summary of some existing neural speech synthesis models and their architectures.

WaveNet [130] is a generative neural network that is trained end-to-end to model quantized audio waveforms. The model is fully probabilistic and autoregressive, using a stack of causal convolutional layers to condition the predictive distribution for each audio sample on all previous ones. It has produced impressive results for generation of speech audio conditioned on speaker and text and has become a standard baseline for neural speech generative models.

SampleRNN [92] is another autoregressive architecture that has been successfully used to generate both speech and music samples. SampleRNN uses a hierarchical structure of deep RNNs to model dependencies in the sample sequence. Each deep RNN operates at a different temporal resolution so as to model both long term and short term dependencies.

Recent work on deep learning architectures has also introduced the presence of *adversarial examples*: small perturbations to the original inputs, normally imperceptible to humans, which nevertheless cause the architecture to generate an incorrect or deliberately chosen output. In their brilliant papers, [127] and [56] analyze the origin of adversarial attacks and describe simple and very efficient techniques for creating such perturbations, such as the fast gradient sign method (FGSM).

In the vision domain, [121] describe a technique for attacking facial recognition systems. Their attacks are physically realizable and inconspicuous, allowing an attacker to impersonate another individual. In the speech domain, [21] describe attacks on speech-recognition systems which use sounds that are hard to recognize by humans but interpreted as specific commands by speech-recognition systems.

[1] We use the term fake to refer to computer generated samples

To the best of our knowledge, GANs have not been used for the purpose of speech synthesis[2]. [103] uses a conditional GAN for the purpose of speech *enhancement*, i.e. taking as input a raw speech signal and outputting a denoised waveform. The model in [23] tackles the reverse problem of using GANs to learn certain representations given a speech spectrogram.

4.3 Attacking Text-Independent Speaker Recognition Systems

In this section, we define the neural speaker recognition system used in our experiments and define the targeted and untargeted adversarial attacks we investigate.

4.3.1 Neural speaker recognition system

The speaker recognition system used in our experiments is based on the framework by [85] and is described in Figure 4.1. The first module at the bottom is a pre-processing step that extracts the Mel-Spectrogram from the waveform as described in section 4.4.2. The second module is a convolutional neural network (CNN) that performs multi-speaker classification using the Mel-Spectrogram. The CNN is a modified version of Alexnet [74]. We warn the readers that unlike [85], our classifier operates on 64 by 64 Mel-Spectrogram and has slightly different number of nodes on each layer.

We train our speaker classifier using 64 by 64 Mel-Spectrograms [3] from 3 speech datasets, including 100 speakers from NIST 2004, speaker p2280 from CSTR VCTK and the single speaker in Blizzard. Our speaker classifier has a rejection path, the "other" class, trained on environmental sounds using samples from the ESC-50 dataset. Our model achieves approximately 85% test set accuracy

4.3.2 Adversarial attacks

We define adversarial attacks on speaker recognition systems as *targeted* or *untargeted*. In targeted attacks, an adversary is interested in designing an input that makes the classification system predict a target class chosen by the adversary. In untargeted attacks, the adversary is interested in a confident prediction, regardless of the class being predicted as long as it is not a "other" class. Untargeted attacks are essentially designed to fool the classifier into thinking a fake speech sample is real. Notice that a successful targeted attack is by definition a successful untargeted attack as well.

[2] More specifically, Mel-Spectrogram synthesis

[3] 64 mel bands and 64 frames, 100 ms each

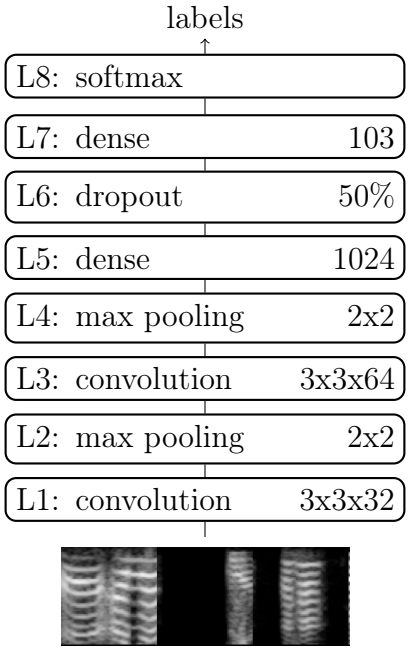


Figure 4.1: Architecture for CNN speaker verifier.

4.4 Experimental Setup

In this section we describe the datasets used and the data engineering pipeline, including pre-processing and feature extraction.

4.4.1 Datasets

In our experiments we use three speech datasets and one dataset with environmental sounds, as shown in Table 4.1. The datasets used are public and provide audio clips of different lengths, quality, language and content. In addition to the samples listed in Table 4.1, we used globally conditioned sampleRNN and WaveNet fake samples available on the web. The samples generated with sampleRNN and WaveNet are from the Blizzard dataset and CSTR VCTK (P280) respectively. We warn the reader that although most of the WaveNet and SampleRNN samples we used carry the timbre of the speaker, the sentences they produce are equal to *babble*.

4.4.2 Pre-processing

Data pre-processing is dependent on the model being trained. For SampleRNN and WaveNet, the raw audio is reduced to 16kHz and quantized using the μ-law

	Speakers	Language	Duration	Context
2013 Blizzard	1	English	73 h	Book narration
CSTR VCTK	109	English	400 Sentences	Newspaper narration
2004 NIST	100	Multiple	5 min / speaker	Conversational phone speech.
ESC 50	50	N/A	4 min / class	Environmental sounds.

Table 4.1: Description of the datasets used in our experiments.

companding transformation as referenced in [92] and [130]. For the model based on the Wasserstein GAN, we pre-process the data by converting it to 16kHz and removing silences by using the WebRTC Voice Activity Detector (VAD) as referenced in [141]. For the CNN speaker recognition system, the data is pre-processed by resampling to 16kHz when necessary and removing silences by using the aforementioned VAD.

4.4.3 Feature extraction

SampleRNN and WaveNet operate at the sample level, i.e. waveform, thus requiring no feature extraction. The features used for the neural speaker recognition system are based on Mel-Spectrograms with dynamic range compression. The Mel-Spectrogram is obtained by projecting a spectrogram onto a mel scale. We use the python library librosa [91] to project the spectrogram onto 64 mel bands, with window size equal to 1024 samples and hop size equal to 160 samples, i.e. 100ms long frames. Dynamic range compression is computed as described in [85], with $log(1 + C * M)$, where C is a compression constant scalar set to 1000 and M is a matrix representing the Mel-Spectrogram. Training the GAN is also done with Mel-Spectrograms of 64 bands and 64 frames image patch.

4.4.4 Models

4.4.4.1 WaveNet

Due to constraints on computing power and the extreme difficulty in training WaveNet [4], we used samples from WaveNet models that had been pre-trained for 88 thousand iterations. Parameters of the models were kept the same as those in [130]. The ability of WaveNet to perform *untargeted* attacks amounts to using a model trained on an entire corpus. Targeted attacks are more difficult - we found that a single speaker's data was not enough to train WaveNet to converge successfully. To construct speaker-dependent samples, we relied on samples from pre-trained models that were *globally conditioned* on speaker ID. Based on informal listening experiments, such samples do sound very similar to the real speech of the speaker in question.

[4]Our community has not been able to replicate the results in Google's WaveNet paper

4.4.4.2 sampleRNN

Similarly to WaveNet, we found that the best (least noisy) sampleRNN samples came from models which were pretrained with a high number of iterations. Accordingly, we obtained samples from the three-tiered architecture, trained on the Blizzard 2013 dataset [107], which as mentioned in Section 3 is a 300 hour corpus of a single female speaker's narration. We also downloaded samples from online repositories, including samples from the original paper's online repository at `https://soundcloud.com/samplernn/sets`, which we qualitatively found to have less noise than our generated ones.

4.4.4.3 WGAN

In all of our experiments, we use the Wasserstein GAN with this gradient penalty (WGAN-GP), which we found makes the model converge better than regular WGAN or GAN. We will henceforth use WGAN, IWGAN, GAN, and WGAN-GP interchangeably to refer to WGAN-GP.

In our experiments, we trained a WGAN-GP to produce mel-spectrograms from 1 target speaker against a set of 101 speakers. On each critic iteration, we fed it with a batch of samples from one target speaker, and a batch of data uniformly sampled from the other speakers.
We used two popular architectures for generator/critic pairs:

- *DCGAN* [108] models the generator as a series of deconvolutional layers with ReLU activations, and the discriminator as a series of convolutional layers with leaky ReLU activations. Both architectures use batch normalization after each layer.

- *ResNet* [79] models the generator and discriminator each as very deep convnets (30 layers in our experiments) with upsampling/downsampling respectively. Residual (skip) connections are added every few layers to make training easier.

Performing *untargeted* attacks with the WGAN-GP (i.e., training the network to output speech samples that mimic the distribution of speech) is relatively straightforward: we simply train the WGAN-GP using all speakers in our dataset. However, the most natural attack is one that is *targeted*: where the GAN is trained to directly fool a speaker recognition system, i.e., to produce samples that the system classifies as matching a target speaker with reasonable confidence.

4.4.4.4 WGAN-GP with modified objective function

A naive approach for targeted attacks is to train the GAN on the data of the single target speaker. A drawback of this approach is that the *critic*, and by consequence

the *generator*, does not have access to universal properties of speech[5]. To circumvent this problem, we rely on unsupervised learning and propose a modification to the critic's objective function that allows it to learn to differentiate between not only real samples and generated samples, but also between real speech samples from a target speaker and real speech samples from other speakers. We do this by adding a term to the critic's loss that encourages its discriminator to classify real speech samples from untargeted speakers as fake. From equation 2.9, the critic's loss L_C changes to:

$$\underbrace{\mathop{\mathbb{E}}_{\widetilde{\boldsymbol{x}}\sim\mathbb{P}_g}\left[D(\widetilde{\boldsymbol{x}})\right]}_{\text{Generated Samples}} + \alpha * \underbrace{\mathop{\mathbb{E}}_{\dot{\boldsymbol{x}}\sim\mathbb{P}_{\dot{x}}}\left[D(\dot{\boldsymbol{x}})\right]}_{\text{Different Speakers}} - \underbrace{\mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}\left[D(\boldsymbol{x})\right]}_{\text{Real Speaker}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{x}}}\left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right]}_{\text{Gradient Penalty}}, \quad (4.1)$$

where $P_{\hat{x}}$ is the distribution of samples from other speakers and α is a tunable scaling factor. Note that equation 4.1 is no longer a direct approximation of the Wasserstein distance. Rather, it provides a balance of the distance between both the fake distribution and real one, and the distance between other speakers' distribution and the target speaker's one. We refer to this objective function as **mixed loss**.

Initially, we were able to converge the targeted loss model used the same parameters as [59], namely 5 critic iterations per generator iteration, a gradient penalty weight of 10, and batch size of 64. Both the generator and critic were trained using the Adam optimizer [71]. However, under these parameters we found that the highest α weight we could successfully use was 0.1 (we found that not including this scaling factor led to serious overfitting and poor convergence of the GAN).

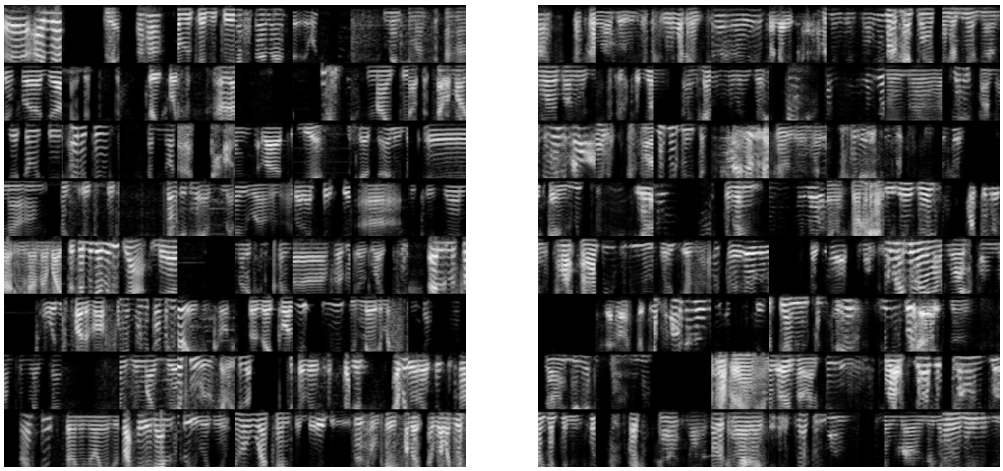
In order to circumvent these problems and train a model with α set to 1, we made modifications to the setup, including setting the standard deviation of the DCGAN discriminator's weight initialization to 0.05 and iterations to 20. To accommodate the critic's access to additional data in the mixed loss function (4), we increased the generator's learning rate. Finally, we added Gaussian noise to the target speaker data to prevent overfitting.

4.5 Experimental Results

4.5.1 GAN Mel-Spectrogram

Using the improved Wasserstein GANs framework, we trained generators to construct 64x64 mel-spectrogram images from a noise vector. Visual results are demonstrated below in Figure 4.2. We saw recognizable Mel-Spectrogram-like features in the data after only 1000 generator iterations, and after 5000 iterations the generated samples were indistinguishable from real ones. Training took around 10 hours for 20000 iterations on a single 4 GB Nvidia GK104GL GPU.

[5]We draw a parallel with Universal Background Models in speech.

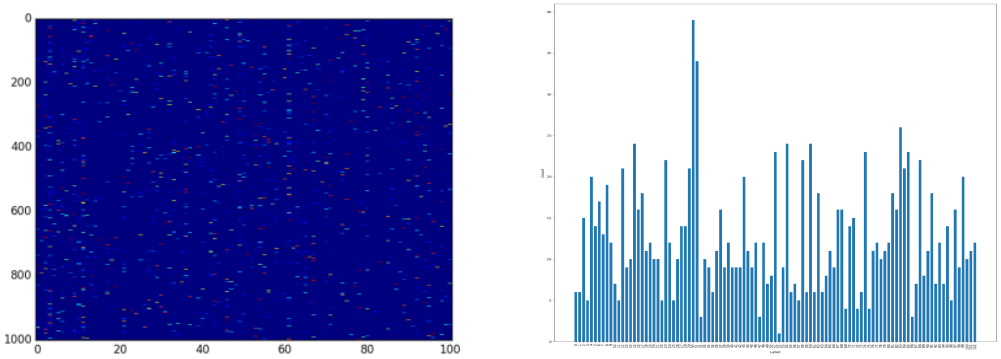


(a) Real (actual) (b) Fake (generated)

Figure 4.2: Comparison of real and generated ($\sim$ 5000 generator iterations) spectrogram samples from all speakers. Each grid contains 64 samples.

4.5.2 GAN Adversarial attacks

Within the GAN framework, we train models for untargeted attacks by using all data available from speakers that the speaker recognition systems was trained on, irrespective of class label. We show that an untargeted model able to generate data from the real distribution with enough variety can be used to perform adversarial attacks. We provide details in the untargeted attacks subsection 4.5.2.1. Figure 4.3b depicts that our GAN-trained generator successfully learns all speakers across the dataset, without mode collapsing.



(a) Our speaker classifier's softmax distribution of 1000 samples on approximately 100 speakers.

(b) Our speaker classifier's distribution of randomly sampled speech from the generative model.

Figure 4.3: Summary of untargeted attacks. Red represents high confidence.

Normally, models for targeted attacks with GANs are trained in two manners:

1. conditioning the models on additional information [95] such as class labels;
2. using only data from the target class;

While the first approach might result in mode collapse, a drawback of the second approach is that the discriminator, and by consequence the generator, do not have access to universal[6] properties of speech. In the targeted attacks subsection 4.5.2.2 we show results using our new *semi-supervised objective function* described in equation 4.1 that uses data from all classes and recognizes and produces data from a single class.

4.5.2.1 Untargeted attacks

For each speaker audio data in the test set, we compute a Mel-Spectrogram as described in section 4.4.2. The resulting Mel-Spectrogram is then fed into the CNN recognizer and we extract a 1024-dimensional feature Φ from the first fully-connected layer (L5) in the pre-trained CNN model (4.1) trained on the real speech dataset with all speaker IDs. This deep feature/embedding Φ is then used to train a K-nearest-neighbor (KNN) classifier, with K equal to 5.

To obtain class labels from samples produced with our generator, we feed the generated Mel-Spectrograms into the same CNN-L7 pipeline to extract their corresponding feature $\widehat{\Phi}$. Utilizing the pre-trained KNN, each sample is assigned to the nearest speaker in the deep feature space. Therefore, we know which speaker our generated sample belongs to when we attack our CNN recognizer. We evaluate our controlled WGAN-GP samples against our CNN speaker recognition system and provide a confusion matrix in Figure 4.4.

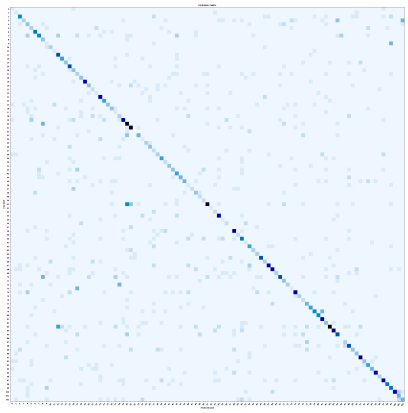


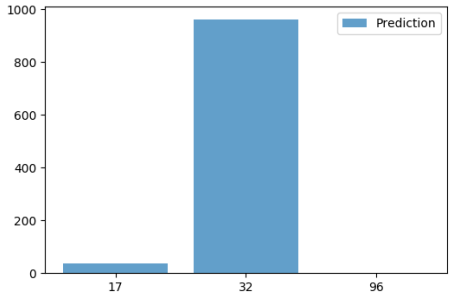
Figure 4.4: Confusion matrix of GAN untargeted attacks. x-axis corresponds to predicted label, y-axis to ground truth.

[6] We draw a parallel with Universal Background Models in speech recognition.

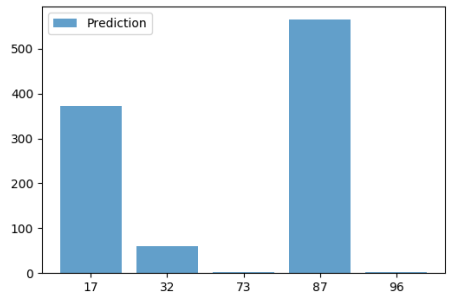
4.5.2.2 Targeted attacks

We ran all three models (WGAN-GP, SampleRNN, WaveNet) on a mixed corpus containing the entirety of the NIST 2004 corpus, a single speaker (P280) from the VCTK Corpus, and the single speaker from the Blizzard dataset. The mixed corpus therefore contains 102 speakers. Samples were either downloaded from the web or created from WaveNet globally conditioned on the single VCTK corpus speaker, and on SampleRNN trained only on data from the Blizzard dataset.

Results on WaveNet and SampleRNN are demonstrated in Figure ??. Neither WaveNet samples nor sampleRNN samples were able to attack the speaker recognition model. In the sampleRNN and WaveNet models, all the predictions made by the classifier match a speaker class but **none** of the predictions matches the target speaker.



(a) Histogram of predictions on sampleRNN data. Target label: 100.



(b) Histogram of predictions on WaveNet data. Target label: 101.

We also trained the WGAN-GP with and without our **mixed loss** on speaker 0. The histogram of predictions in Figure 4.6 shows that with the mixed loss model, most of the energy is concentrated on the target speaker 0. Our mixed loss achieves achieves 12% error: a 75% relative increase in accuracy over naive WGAN-GP.

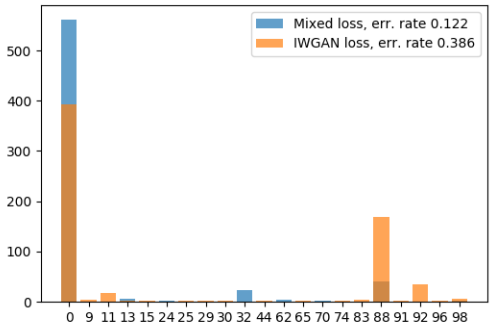


Figure 4.6: Histogram of predictions on IWGAN and mixed loss data. Target label: 0.

It is therefore clear that in this context the WGAN-GP mixed loss is an improvement over the original loss function. This is expected given the network's access to additional speaker data.

4.6 Discussion

In this research we have investigated the use of speech generative models to perform adversarial attacks on speaker recognition systems. We show that the samples from autoregressive models we trained, i.e. SampleRNN and WaveNet, or downloaded from the web were not able to fool the CNN speaker recognizers we used in this research. On the other hand, we show that adversarial examples generated with GAN networks are successful in performing targeted and untargeted adversarial attacks given the speaker recognition system used in our experiments.

It is important to mention that the generative models used in this research have different loss functions: whereas WaveNet and sampleRNN learn next-step prediction, the GAN framework trains a generator to fool a discriminator by producing fake samples that the discriminator believes to come from the same distribution as the real samples. The loss of both training schemes is very different from the loss used in training a speaker recognizer and none of the generative models, including GANs, interact with the speaker recognition system. Also worth mentioning is that the GAN generator we used is unconditional an incapable of generating samples based on previous samples.

Although we have ongoing successful experiments on training sequential GANs to produce samples conditioned on previous samples, fooling text-independent speaker classifiers is an important intermediate step to evaluate the ability of speech generative models in their capacity to fool speaker classifiers. It is reasonable to affirm that a generative model that does not fool a text-independent speaker classifier, like WaveNet and sampleRNN does not, will not fool a text-dependent speaker classifier. Not reported in this paper, the generative models were also tested against a GMM-UBM i-vector speaker classifier trained using Alize and WaveNet and SampleRNN did not fool the GMM-UBM classifier.

A pertinent argument against the validity of the informal GMM-UBM tests lies on the fact that GMM-UBM models have high precision and would not generalize to speech in different conditions, e.g. different room or microphone. First, it is not within the scope of this research to build a speaker recognition system that is invariant to room and microphone conditions. Second, given that the speaker recognition models, CNN and informally GMM-UBM, have good performance on test data and that WaveNet and SampleRNN goal is to replicate speech data that is from a speaker with similar and fixed microphone and room conditions, it is expected that the outputs of these generative models should be properly classified by the speaker recognition system.

With this research we hope to raise attention to issues that generative models bring to security and biometric systems. We foresee that samples produced with generative models have signatures that can be used to identify the source of the data. This hypothesis is investigated in the next chapter.

Chapter 5

Interesting properties of samples generated with Generative Adversarial Networks

5.1 Introduction and Motivation

Since the groundbreaking Generative Adversarial Networks paper [55] in 2014, most GAN related publications use a grid of image samples to accompany theoretical and empirical results. Given this context, the expansion of GAN research to other domains including language models [59] and music [139] display the need of sample inspection.

Unlike Variational Autoencoders (VAEs) and other models [55], most of the evaluation of the output of Generators trained with the GAN framework is qualitative: authors normally list higher sample quality as one of the advantages of their method over other methods. Interestingly, little is mentioned about the numerical properties of GAN samples and how these properties compare to real samples.

In the context of Verified Artificial Intelligence[120], it is hard to systematically verify the Generator and the samples it produces because verification might depend on the existence of perceptually meaningful features. For example, consider the generation of images of humans: although it is possible to compare color histograms of real and fake[1] samples, we do not yet have robust algorithms able to verify if an image follows specifications derived from anatomy.

This research is related to this systematic sample verification and focuses on understanding the numerical properties of GAN samples. We investigate how the Generator approximates modes in the real distribution and verify if the generated samples violate specifications derived from the real distribution. We offer the following contributions in this research:

[1] Generated samples

- We show that GAN samples have universal signatures.
- We show how GAN samples approximate modes of the real distribution.
- We show significant differences between the marginal distribution of features.
- We show GAN samples that violate specifications in the real data.

5.2 Hypotheses

Hypothesis 1 (H1)**.** *Generative models can approximate the distribution of real data and hallucinate fake data that has some variety and resembles real data.*

Although this hypothesis is trivial for experiments that have already been conducted, it is the first condition for our experiments. To our knowledge there are no publications where GANs are successful in hallucinating polyphonic music and speech data. During our experiments we prove that these hypotheses hold.

Hypothesis 2 (H2)**.** *The real data has useful properties that can be extracted computationally.*

By useful we refer to properties that can be used to describe specifications of the real data. For example, computing the distribution MNIST pixel values might be not useful for assessing drawing quality but it might be useful to evaluate if a random MNIST sample is real or fake.

Hypothesis 3 (H3)**.** *The fake data has properties that differ from the real data and are hardly noticed with visual inspection of samples.*

Visual inspection of generated samples has become the norm for the evaluation of samples generated using the GAN framework. We investigate if there are properties common to all GAN samples or properties that significantly differ between the real data and the fake data. This hypothesis supports the next hypothesis related to adversarial attacks.

Hypothesis 4 (H4)**.** *The difference in properties can be used to identify the source (real or fake)*

The development of generative models foreshadows the imminent rise of adversarial attacks. We investigate if these differences can be used to detect the source of the data (real, GAN or adversarial attack).

With respect to hypotheses 2 and 4, we call the reader's attention that approximating the distribution over features computed on the real data does not guarantee that the real data is being approximated. Formally speaking: consider $X \sim Z$, i.e. X distributed as Z, and $f(X) \sim W$, where $f : X \mapsto Y$. If $A \sim B$ and B approximates Z, then $f(A) \sim D$ must also approximate W. However, a distribution that approximates W is not guaranteed to approximate Z.

5.3 Experimental Setup

In this section we describe our methodology, briefly describing the datasets and features computed, as well as the model architectures and GAN algorithms used.

5.3.1 Datasets

In our experiments, we use the MNIST dataset, a MIDI dataset of 389 Bach Chorales downloaded from the web and a subsample of the NIST 2004 telephone conversational speech dataset with 100 speakers, multiple languages and on average 5 minutes of audio per speaker.

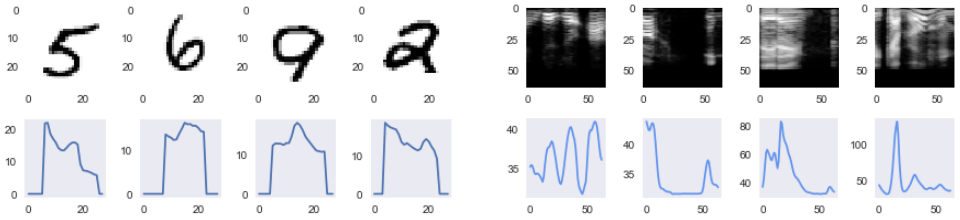
5.3.2 Property extraction

The properties extracted from the datasets used on this research can be perceptually meaningful or not. We claim that both properties can be used to numerically identify the source of the sample. In the context of this paper, samples are images of size 64 by 64.

5.3.2.1 Spectral Moments

The spectral centroid [105] is a feature commonly used in the audio domain, where it represents the barycenter of the spectrum. This feature can be applied to other domains and we invite the reader to visualize Figure 5.1 for examples on MNIST and Mel-Spectrograms [105]. For each column in an image, we transform the pixel values into row probabilities by normalizing them by the column sum, after which we take the expected row value, thus obtaining the spectral centroid.

Figure 5.1a shows the spectral centroid computed on sample of MNIST training data.

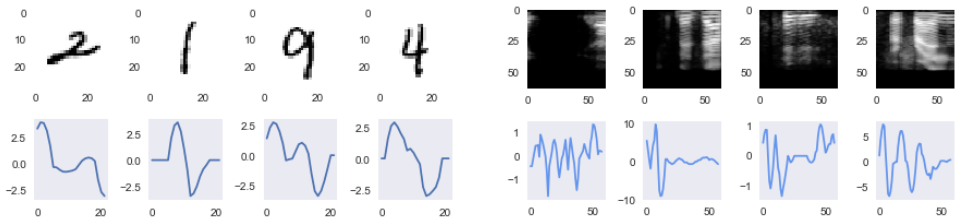


(a) MNIST samples and centroids (b) Mel-Spectrograms and centroids

Figure 5.1: Spectral centroids on digits and Mel-Spectrograms

5.3.2.2 Spectral Slope

The spectral slope adapted from [105] is computed by applying linear regression using an overlapping sliding window of size 7. For each window, we regress the spectral centroids on the column number *mod* the window size. Figure 5.2 shows these features computed on MNIST and Mel-Spectrograms from the NIST dataset.



(a) MNIST samples and slopes (b) Mel-Spectrograms and slopes

Figure 5.2: Spectral slopes on digits and Mel-Spectrograms

5.3.3 Generative Models

We investigate samples produced with the DCGAN architecture using the Least-Squares GAN (LSGAN) [88] and the improved Wasserstein GAN (IWGAN) [59]. We also compare adversarial MNIST samples produced with the fast gradient sign method (FGSM) [56].

5.3.4 Distance Measures

We use the Jensen-Shannon divergence and the Kolgomorov-Smirnov Two-Sample test [89] for comparing distributions. The Jensen-Shannon divergence is described in 2.5 and below we provide the equation describing the Kolgomorov-Smirnov Two-Sample test:

$$D_{n,m} = \sup_x |F_{1,n}(x) - F_{2,m}(x)|, \tag{5.1}$$

where $F_{1,n}$ and $F_{2,m}$ are the empirical cumulative distribution functions (eCDF) of the first and the second sample respectively, and sup is the supremum function. The null hypothesis is rejected at level α if:

$$D_{n,m} > c(\alpha)\sqrt{\frac{n+m}{nm}}, \tag{5.2}$$

where n and m are the sizes of first and second sample respectively. The value of $c(\alpha)$ can be computed with:

$$c(\alpha) = \sqrt{-\frac{1}{2}\ln\left(\frac{\alpha}{2}\right)}. \tag{5.3}$$

5.4 Experimental Results

5.4.1 MNIST

We compare the distribution of features computed over the MNIST training set to other datasets, including the MNIST test set, samples generated with GANs and adversarial samples computed using FGSM. The training data is scaled to $[0, 1]$ and the random baseline is sampled from a Bernoulli distribution with probability equal to the mean value of pixel intensities in the MNIST training data, 0.13. Each GAN model is trained until the loss plateaus and the generated samples look similar to the real samples. The datasets compared have 10 thousand samples each.

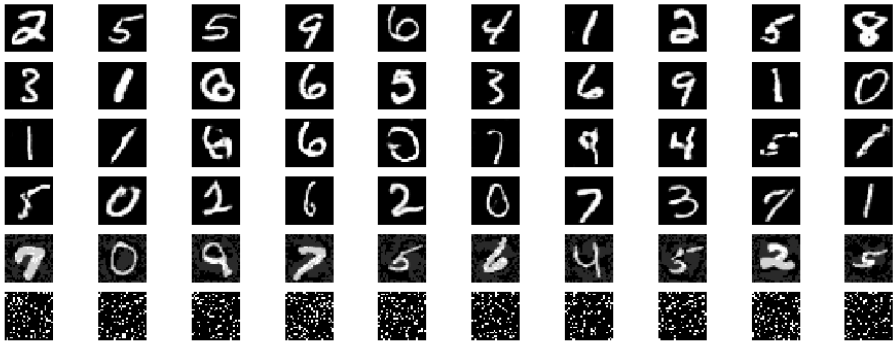


Figure 5.3: Samples from MNIST train, test, LSGAN, IWGAN, FSGM and Bernoulli.

Visual inspection of the generated samples in Figure 5.3 show that IWGAN seems to produce better samples than LSGAN. Quantitatively, we use the MNIST training set as a reference and compare the distribution of pixel intensities. Table 5.1 reveals that although samples generated with LSGAN and IWGAN look similar to the training set, they are considerably different from the training set given the Kolgomorov-Smirnov (KS) Two Sample Test and the Jensen-Shannon Divergence (JSD), specially if compared to the same statistics on the MNIST test data.

	KS Two Sample Test		JSD
	Statistic	P-Value	
mnist_train	0.0	1.0	0.0
mnist_test	0.003177	0.0	0.000029
mnist_lsgan	0.808119	0.0	0.013517
mnist_iwgan	0.701573	0.0	0.014662
mnist_adversarial	0.419338	0.0	0.581769
mnist_bernoulli	0.130855	0.0	0.0785009

Table 5.1: Statistical comparison over the distribution of pixel values for different samples using MNIST training set as reference.

These numerical phenomena can be understood by investigating the empirical CDFs in Figure 5.4. The pixel values of the samples generated with the GAN framework are mainly bi-modal and smoothly and asymptotically approach the modes of the distribution of pixel values in the real data, 0 and 1.

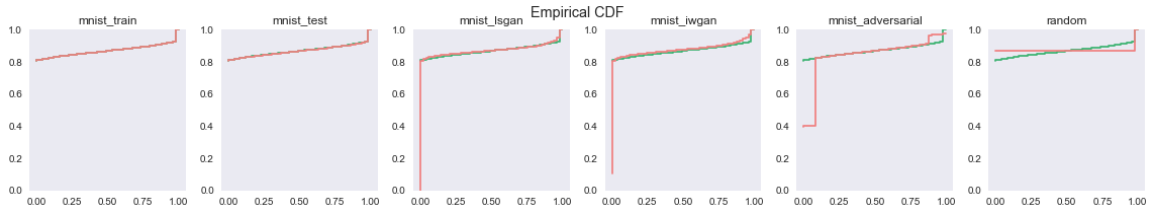


Figure 5.4: Pixel empirical CDF of training data as reference (green) and other datasets(red)

In addition, a plot of the distribution of moments of the spectral centroid in Figure 5.5 suggests that the fake images are more noisy than the real images. Consider for example the images produced by randomly sampling a Bernoulli distribution with parameter estimated from the training data. These images have pixel values of zero and one that are equally distributed [2] over the image. Well, an image that has pixels values distributed in such manner will have distribution of mean spectral centroid that will be centered at the center row of the image, row 14. This and the similarity between the distribution of mean spectral centroids from fake data and Bernoulli data shows evidence that the fake images have noise that are also equally spatially distributed.

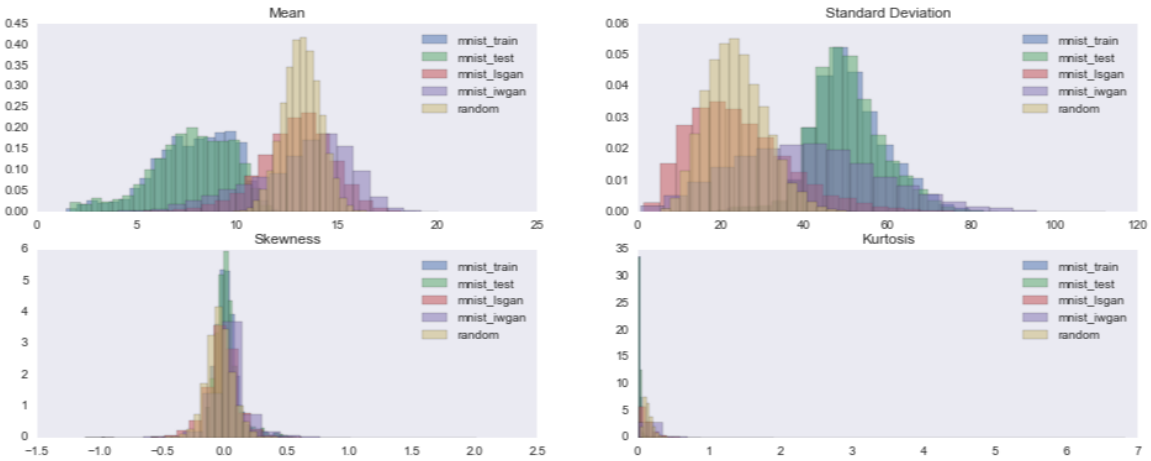


Figure 5.5: Distribution of moments of spectral centroids computed on each image.

Last, Figure 5.6 shows that the GAN generated samples smoothly approximate the modes of the distribution. This smooth approximation is considerably different

[2] This spatial distribution is independent of the parameter of the Bernoulli distribution.

from the training and test sets. Although these properties are not perceptually meaningful, they can be used to identify the source of the data, hence confirming hypotheses 2, 3 and 4.

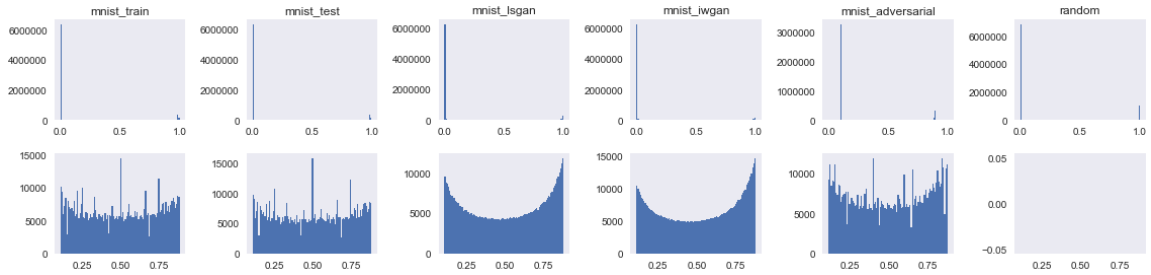
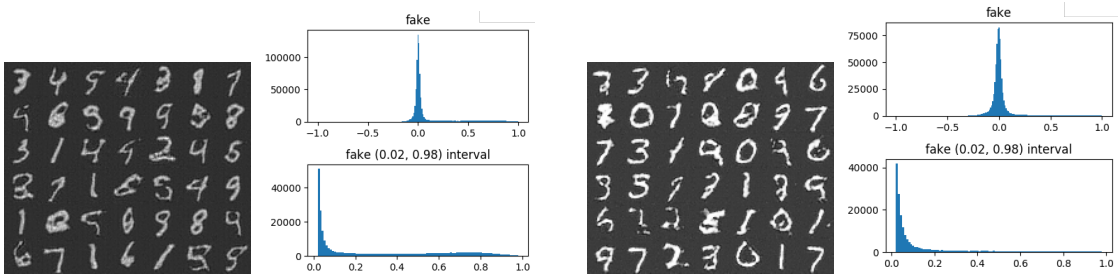


Figure 5.6: Histogram of pixel intensities for each dataset. First row shows histogram within the [0, 1] interval and 100 bins. Second row shows histograms between the [0.11, and 0.88] interval and 100 bins.

Our first hypothesis for the smooth approximation of the modes of the distribution was that it would be present in any data produced with a generator that is trained using stochastic gradient descent and an asymptotically converging activation function, such as sigmoid or tanh, at the output of the generator. To evaluate this hypothesis, we conducted a set of experiments using different GAN architectures (WGAN-GP, LSGAN, DCGAN, LSGAN) with different activation functions, including linear and the scaled tanh, at the output of the Generator, keeping the discriminator fixed.

To our surprise, we noticed that the models trained with linear or scaled tanh activations were not able to produce images that were similar to the MNIST training data and, to our surprise, the distribution of pixel intensities, although uni-modal around zero, still possessed a smooth looking curve. This is illustrated in Figure 5.7.

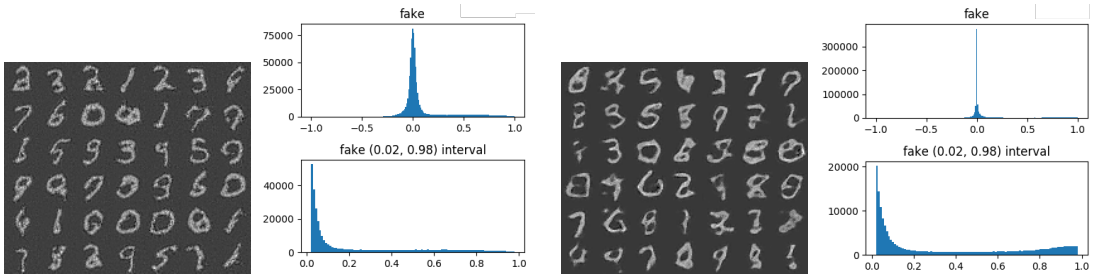


(a) Linear (b) Scaled Tanh

Figure 5.7: Fake MNIST samples and pixel distribution from generators trained with DCGAN, Batch Norm and linear or scaled tanh activation functions.

We then hypothesized that the smooth behavior was due to smoothness in the pixels intensities of the training data itself. To validate this hypothesis, we binarized

the real data by first scaling it between $[0, 1]$ and then thresholding it at 0.5. With this alteration the distribution of the pixel intensities of the real data becomes completely bi-modal with modes at 0 and 1. Interestingly, even with binarized training data, the smooth behavior was still present as we show in Figure 5.8.



(a) DCGAN Linear, Binary samples (b) WGAN-GP Linear, Binary samples

Figure 5.8: Fake MNIST samples and pixel distribution from generators trained on binarized real data with DCGAN and WGAN-GP, Batch Norm and linear activation functions.

With this empirical evidence at hand, we provide an informal analysis of the smoothness of the distribution of pixels of the generated data from the perspective of optimization, differentiation and function approximation with neural networks. We know that backpropagation and stochastic gradient descent are used to update the weights of a neural network model based on the gradient of the loss with respect to the weights. We also know that differentiation requires the function that is being differentiated to have some high degree of smoothness and be differentiable almost everywhere [3]. Hence, we conjecture that the inductive bias of the neural networks is that of smoothness given the requirements of differentiation.

We speculate that this inductive bias is responsible for the smoothness of the distribution of pixel values at any iteration during training and that the U shape of the distribution of pixel values, similar to blurring, is the byproduct of an smooth approximation of the function that is being learned [4]. We forward the interested user to our github repo [5] in which we provide code to replicate our experiments. This behavior probably explains why the smooth behavior is also present on the GAN generated Bach chorales studied in the next experiment.

5.4.2 Bach Chorales

We investigate the properties of Bach chorales generated with the GAN framework and verify if they satisfy musical specifications. Bach chorales are polyphonic pieces

[3] ReLU is not smooth at 0 for example but works fine.

[4] Consider approximating a function with polynomials of increasing degrees

[5] `https://github.com/rafaelvalle/ipgans`

of music, normally written for 4 or 5 voices, that follow a set of specifications/rules[6]. For example, a global specification could assert that only a set of durations are valid; a local specification could assert that only certain transitions between states (notes) are valid depending on the current harmony.

For this experiment, we convert the dataset of Bach chorales to piano rolls. The piano roll is a representation in which the rows represent note numbers, the columns represent time steps and the cell values represent note intensities. We compare the distribution of features computed over the training set, test set, GAN generated samples and a random baseline sampled from a Bernoulli distribution with probability equal to the normalized mean value of intensities in the training data. After scaling, the intensities in the training and test data are strictly bi-modal and equal to 0 or 1. Figure 5.9 below shows training, test, IWGAN and Bernoulli samples, thus confirming hypothesis 1. Each dataset has roughly 1000 image patches and the image patches from training and test data are randomly sampled from the Bach Chorales dataset in piano roll format.

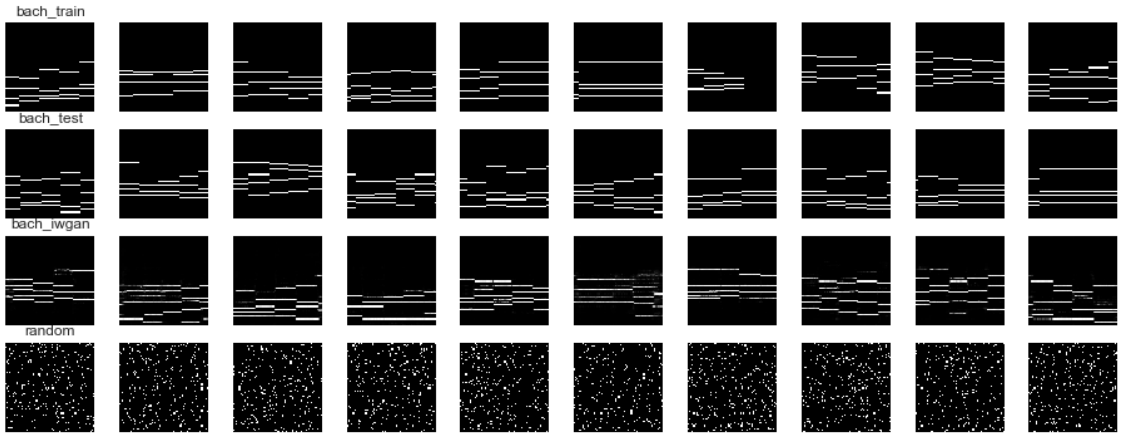


Figure 5.9: Samples drawn from Bach Chorales train, test, IWGAN, and Bernoulli respectively.

Figure 5.10 shows a behavior that is similar to our previous MNIST experiments: the IWGAN asymptotically approximates the modes of the distribution of intensity values. In the interest of space, we refer the reader to the online appendix[7] for statistics and other relevant information.

Following, we investigate if the generated samples violate the specifications of Bach chorales. Given that the intensities of notes in the Bach chorales are strictly bi-modal, we must post-process the GAN samples to define what consists of a silence

[6]The specifications define the characteristics of the musical style.

[7]Not provided to preserve anonymity

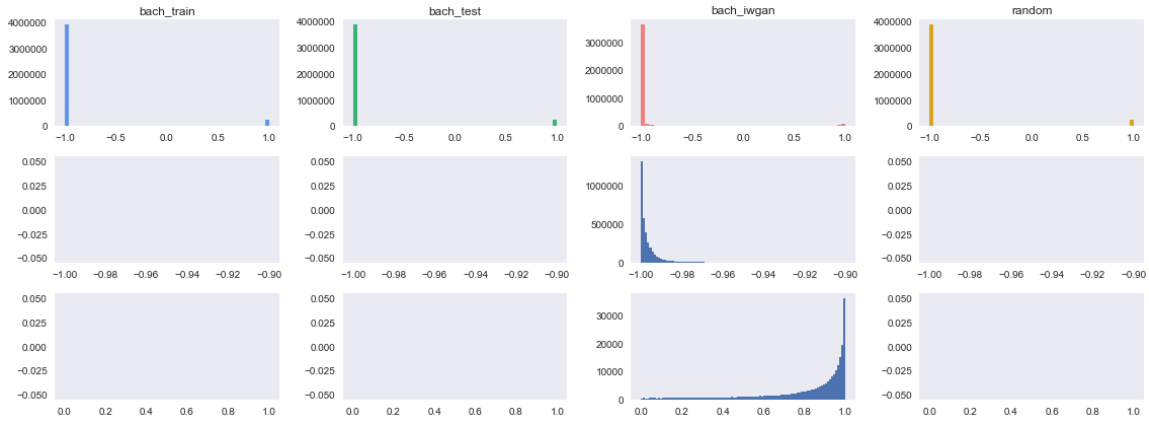


Figure 5.10: Intensity distributions of training, test iwgan and Bernoulli Bach choral samples . Row 1 is in the range $[-1, 1]$, Row 2 is in the range $(-1, 0.9]$ and Row 3 is in the range $[0, 1)$. The figure shows that GAN samples smoothly approximate the modes of the distribution.

or note onset. For doing so, we first convert all datasets to boolean by thresholding at 0.5: values above the threshold are set to 1 or set to 0 otherwise. We use these piano rolls to compute boolean Chroma [105] feature and to compute an empirical Chroma transition matrix, where the positive entries represent existing and valid transitions. The transition matrix built on the training data is taken as the reference specification, i.e. anything that is not included is a violation of the specification. Table 5.2 shows the number of violations given each dataset. Although Figure 5.9 shows generated samples that look similar to the real data, the IWGAN samples have over 5000 violations, 10 times more than the test set! We use these facts to confirm hypotheses 2, 3 and 4.

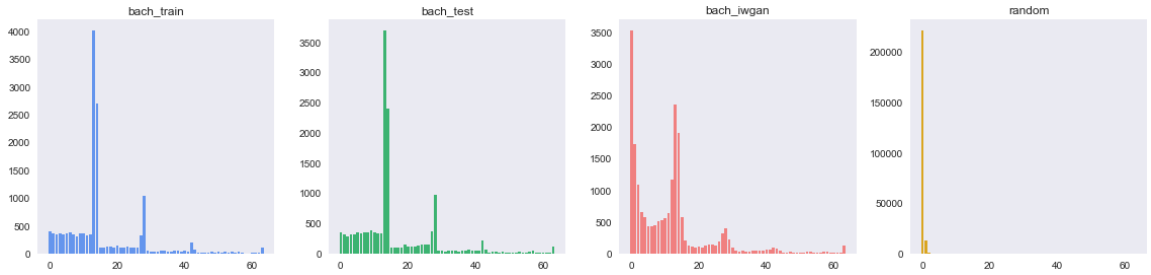
	bach_train	bach_test	bach_iwgan	bach_bernoulli
Number of Violations	0	429	5029	58284

Table 5.2: Number of specification violations with training data as reference.

In addition to experiments with Chroma features, we computed the distribution of note durations on the boolean piano roll described above. Figure 5.11a shows the distribution of note durations within each dataset. The train and test data are approximately bi-modal and, again, the improved WGAN smoothly approximates the dominating modes of the distribution. Table 5.11b provides a numerical comparison between datasets.

5.4.3 Speech

Within the speech domain, we investigate dynamic compressed Mel-Spectrogram samples produced with GANs trained on a subset of the NIST 2004 dataset, with 100



(a) Histogram of note durations

	KS Two Sample Test		JSD
	Statistic	P-Value	
train	0.0	1.0	0.0
test	0.09375	0.929	0.002
iwgan	0.21875	0.080	0.084
bernoulli	0.93750	0.0	0.604

(b) Test statistics

speakers. We divide the NIST 2004 dataset into training and test set, generate samples with the GAN framework and use a random baseline sampled from a Exponential distribution with parameters chosen using heuristics. The generated samples can be seen in Figure 5.12, thus confirming hypothesis 1. We obtain the Mel-Spectrogram by projecting a spectrogram onto a mel scale, which we do with the python library librosa [91]. More specifically, we project the spectrogram onto 64 mel bands, with window size equal to 1024 samples and hop size equal to 160 samples, i.e. frames of 100ms long. Dynamic range compression is computed as described in [85], with $log(1 + C * M)$, where C is the compression constant scalar set to 1000 and M is the matrix representing the Mel-Spectrogram. Each dataset has approximately 1000 image patches and the GAN models are trained using DCGAN with the improved Wasserstein GAN algorithm.

In Figure 5.13a we show the empirical CDFs of intensity values. Unlike our previous experiments where intensity (Bach Chorales) or pixel value (MNIST) was linear, in this experiment intensities are compressed using the log function. This considerably reduces the distance between the empirical CDFs of the training data and GAN samples, specially around the saturating points of the tanh non-linearity, -1 and 1 in this case. In Table 5.13b we show numerical analysis of the differences and confirm hypotheses 2 and 3.

Figure 5.14 shows the distribution of statistical moments computed on spectral centroids and slope. The distributions from different sources considerably overlap, indicating that the generator has efficiently approximated the real distribution of these features.

Figure 5.15 shows statistics used to compare the reference (training data) and

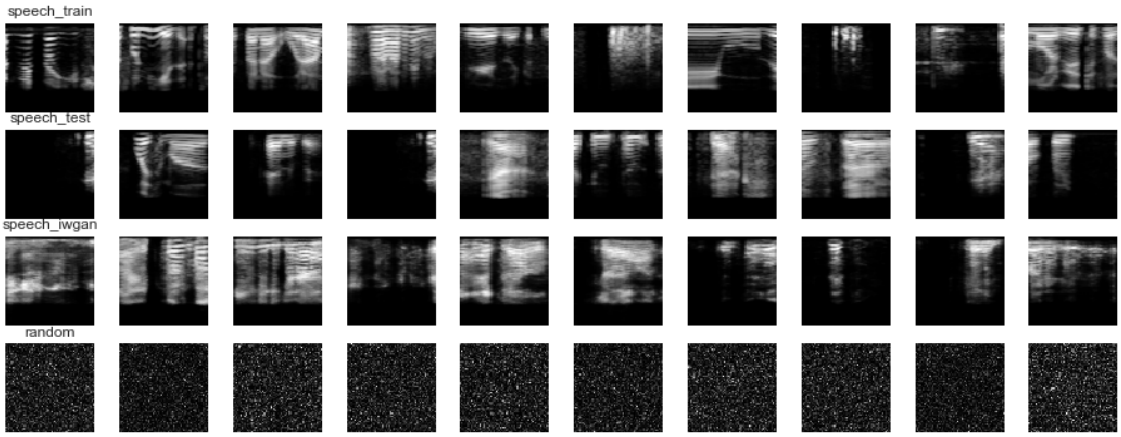
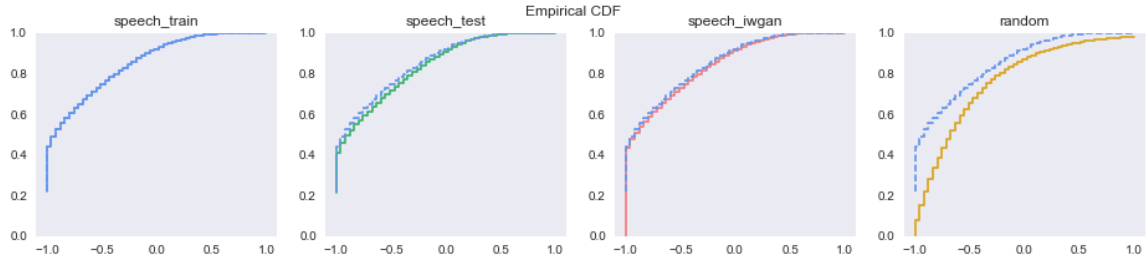


Figure 5.12: Samples drawn from Mel-Spectrogram Speech train, test, IWGAN, and exponential respectively.



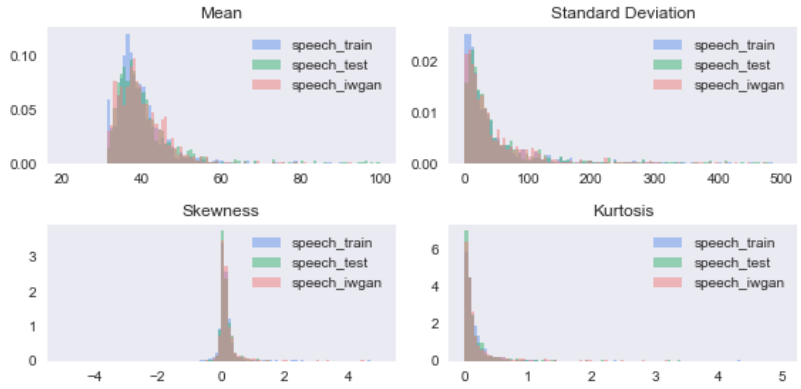
(a) Intensity empirical CDF of training data in blue and other datasets.

	KS Two Sample Test		JSD
	Statistic	P-Value	
train	0.0	1.0	0.0
test	0.03685	0.0	0.00080
iwgan	0.22149	0.0	0.00056
bernoulli	0.36205	0.0	0.11423

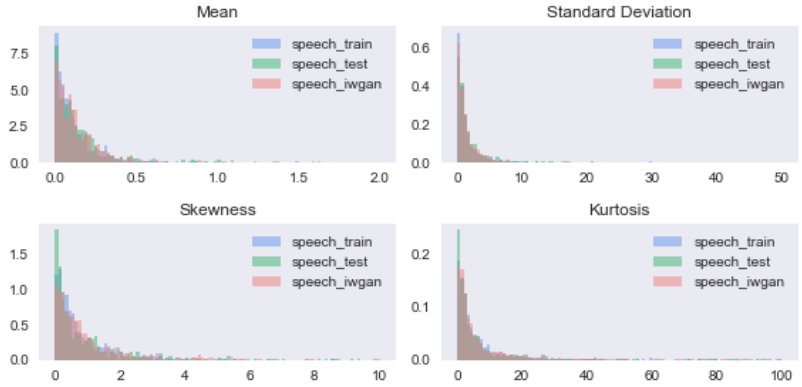
(b) Test statistics

Figure 5.13: Empirical CDF and statistical tests of speech intensity

other datasets. The difference between KS-Statistics and JSD of the test data and generated samples are negligible. Interestingly, the p-values of the spectral slope of the improved WGAN are considerably higher than the test data. For these reasons and although Table 5.13b shows a significant difference between the KS-Statistic of test data and generated data with respect to the training data, we refrain from confirming hypothesis 4. An adversary can easily manipulate the generated data to considerably decrease this difference and still keep the high similarity in features harder to simulate



(a) Spectral Centroid Moments



(b) Spectral Slope Moments

Figure 5.14: Moments of spectral centroid (left) and slope(right)

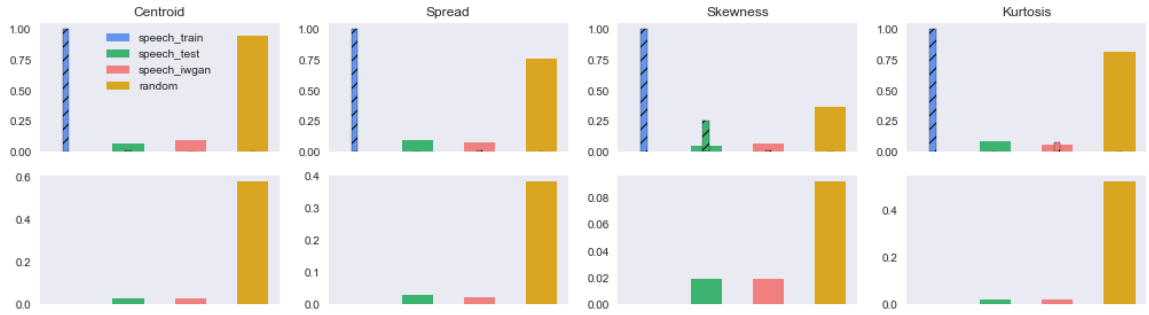
such as moments of spectral centroid or slope.

5.5 Related Work

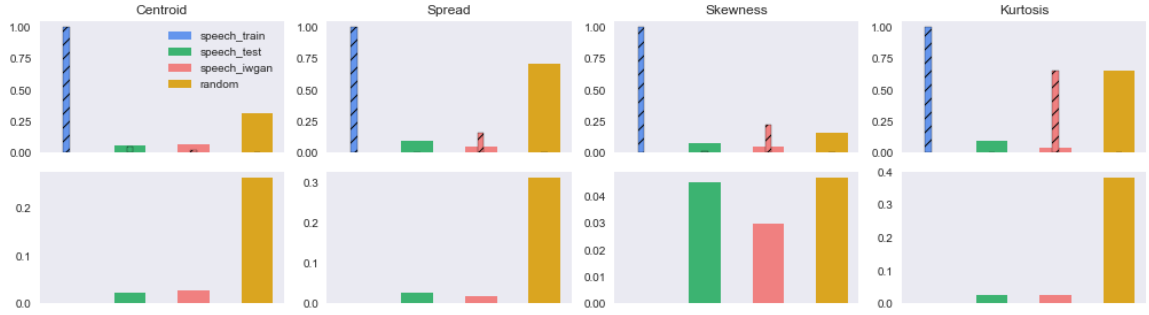
Despite its youth, several publications ([6], [116], [145], [108]) have investigated the use of the GAN framework for generation of samples and unsupervised feature learning. Following the procedure described in [17] and used in [55], earlier GAN papers evaluated the quality of the Generator by fitting a Gaussian Parzen window[8] to the GAN samples and reporting the log-likelihood of the test set under this distribution. It is known that this method has some drawbacks, including its high variance and bad performance in high dimensional spaces [55].

Unlike other optimization problems, where analysis of the empirical risk is a strong indicator of progress, in GANs the decrease in loss is not always correlated

[8]Kernel Density Estimation



(a) Spectral Centroid KS Statistics. First row compares datasets with respect to the distribution of spectral centroid mean computed over each image. Second row compares standard deviation of the same feature.



(b) Spectral Slope KS Statistics. First row compares datasets with respect to the distribution of spectral slopes mean computed over each image. Second row compares standard deviation of the same feature.

Figure 5.15: KS Test-Statistics of spectral centroid (top) and slope (bottom). Pure color is the test Statistics and hatched color is p-value.

with increase in image quality [7], and thus authors still rely on visual inspection of generated images. Based on visual inspection, authors confirm that they have not observed mode collapse or that their framework is robust to mode collapse if some criteria is met ([7], [59], [88], [108]). In practice, github issues where practitioners report mode collapse or not enough variety abound.

In their brilliant publications, [88], [7] and [59] propose alternative objective functions and algorithms that circumvent problems that are common when using the original GAN objective described in [55]. The problems addressed include instability of learning, mode collapse and meaningful loss curves [116].

These alternatives do not eliminate the need or excitement[9] of visually inspecting GAN samples during training, nor do they provide quantitative information about the generated samples. In the following sections, we will analyze GAN samples and reveal some interesting properties therein. In addition to comparing the marginal

[9]Despite of authors promising on twitter to never train GANs again.

distribution of features from the real and fake data, we approach these distributions from the real data as specifications that can be used to validate the output of GAN Samples. We start by enumerating the hypotheses evaluated in this research.

5.6 Discussion

In this research we investigated numerical properties of samples produced with adversarial methods, specially Generative Adversarial Networks. We showed that GAN samples have universal signatures that are dependent on the choice of non-linearity on the last layer of the generator. In addition, we showed that adversarial examples produced with the FSGM have properties that can be used to identify an adversarial attack. Following, we showed that GAN samples smoothly approximate the dominating modes of the distribution and that this information can be used to identify the source of the data. Last, we showed that samples generated with GANs violate specifications and do not provide guarantees on satisfaction of simple specifications. With this we hope to call attention to the necessity of the development of verified AI and better understanding of GAN generated samples.

Chapter 6

Conclusion

In this section we conclude the thesis with a summary of its contributions and future directions.

6.1 Contributions

In this thesis we investigated data hallucination, falsification and validation with generative models and formal methods.

From the perspective of data hallucination, in Chapter 3 we applied the control improvisation framework to the problem of machine improvisation with formal specifications using symbolic music data. First we described an algorithm that uses simple binary patterns to mine specifications in the form of pattern graphs. Then we used the mined specifications to compare data generated with and without the control improvisation framework. Our results showed that showed that the data generated with the framework produces less violations of the specifications, still maintaining artistic interest.

Next, from the perspective of data falsification, in Chapter 4 we evaluated generative models for speech in their efficiency in fooling a time-independent speaker recognition system. We compared the performance of globally conditioned WaveNet, SampleRNN and our GAN models and showed that our spoofing attacks with GANs is considerably more efficient than attacks with the other models, given its low error rate and ease of training. These results were achieved with our modification of the GAN objective function that rewards the model for producing data from the target speaker and penalizes is for producing data from other speakers.

Last, hoping to circumvent adversarial attacks with generated data and understanding samples produced by GANs, in Chapter 5 we studied the properties of samples produced with GANs and compared them to real samples, i.e. the samples that were used to train these GANS. We showed that, although real and fake samples look similar to the bare eye, fake samples produced with GANs or the fast-gradient

sign method (FGSM) have numerical properties that differ from real data and that these numerical properties can be used to identify the source of the data. We show that these differences are also manifested in violations of specification computed over the real data. Finally, we show that data produced with generative models will be an smooth approximation of the real data and conjecture that this smoothness is inherent to differentiability and stochastic gradient descent.

6.2 Future Directions

In this work we combined generative models with formal methods to perform tasks related to data hallucination, falsification and verification. This combination was mainly done from the perspective of mining specifications and using these specifications to generate data in a controlled way or to validate data that was generated. Formal methods can also be combined with neural networks in simple ways, such as adding a penalty term to a generator's loss function that is based on the formal specifications mined from the real data [120], or in more intricate ways to enforce that the neural network is smooth and robust to adversarial attacks. These are extremely hard challenges that have recently been addressed by our community [20,42,68,120] and that we plan to address in the future.

Regarding adversarial attacks in speaker recognition and given that the generative models used in our experiments produced only babble or speech features, it would be valuable to synthesize full speech with these generative models and evaluate their performance on a text-dependent speaker recognition system. Furthermore, following the trend of studies that investigate the properties of data generated with GANs, an analysis of the variety of speech produced by our GANs could show evidence that the GAN models have learned the distribution of the real data, instead of producing a few modes from the distribution.

Artistically, further work in machine improvisation with formal specifications and pattern graphs includes designing an interface that allows a musician to write, mine and combine specifications from MIDI data and visualize pattern graphs extracted from it. In addition to providing artists with a tool for guided improvisation, this provides a new visual tool for music analysis that can shed light onto commonalities and characteristics of large sets of data.

Finally, in our current GAN work, we have trained models that predict the next future block of music given the current block. We are also interested in data generation conditioned on musical abstractions such as the ones used to build pattern graphs in this thesis. This is a point of connection between specification mining and GANs where the mined specifications can be used to condition the output of the GAN generator.

Bibliography

[1] Ilge Akkaya, Daniel J. Fremont, Rafael Valle, Alexandre Donzé, Edward A. Lee, and Sanjit A. Seshia. Control Improvisation with probabilistic temporal specifications. In *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*, pages 187–198. IEEE, 2016.

[2] Rajeev Alur, Pavol Cernỳ, Parthasarathy Madhusudan, and Wonhong Nam. Synthesis of interface specifications for Java classes. *ACM SIGPLAN Notices*, 40(1):98–109, 2005.

[3] Glenn Ammons, Rastislav Bodík, and James R. Larus. Mining specifications. *ACM Sigplan Notices*, 37(1):4–16, 2002.

[4] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. VQA: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2425–2433, 2015.

[5] S. O. Arik, M. Chrzanowski, A. Coates, G. Diamos, A. Gibiansky, Y. Kang, X. Li, J. Miller, A. Ng, J. Raiman, S. Sengupta, and M. Shoeybi. Deep Voice: Real-time neural text-to-speech. *ArXiv e-prints*, February 2017.

[6] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. January 2017.

[7] M. Arjovsky, S. Chintala, and title = Wasserstein GAN journal = arXiv preprint arXiv:1701.07875 year = 2017 Bottou, L.

[8] S. Arora, R. Ge, Y. Liang, T. Ma, and Y. Zhang. Generalization and equilibrium in generative adversarial nets (GANs). *ArXiv e-prints*, March 2017.

[9] S. Arora and Y. Zhang. Do GANs actually learn the distribution? An empirical study. *ArXiv e-prints*, June 2017.

[10] Artificial Intelligence Virtual Artist. Genesis - op. 21 for orchestra, 2017.

[11] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al. The landscape of parallel computing research: a view from Berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[12] Gérard Assayag and Shlomo Dubnov. Using factor oracles for machine improvisation. *Soft Computing*, 8(9):604–610, 2004.

[13] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[14] J. M. Bernardo, M. J. Bayarri, J. O. Berger, A. P. Dawid, D. Heckerman, A. F. M. Smith, and M. West. Generative or discriminative? Getting the best of both worlds. *Bayesian Stat*, 8(3):3–24, 2007.

[15] D. Berthelot, T. Schumm, and L. Metz. BEGAN: Boundary equilibrium generative adversarial networks. *ArXiv e-prints*, March 2017.

[16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End-to-end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[17] Olivier Breuleux, Yoshua Bengio, and Pascal Vincent. Quickly generating representative samples from an RBM-derived process. *Neural Computation*, 23(8):2058–2073, 2011.

[18] Giordano Cabral, Jean-Pierre Briot, and Francois Pachet. Incremental parsing for real-time accompaniment systems. In *The 19th International FLAIRS Conference, Special Track: Artificial Intelligence in Music and Art*. Florida Artificial Intelligence Research Society, 2006.

[19] Michel Caplain. Finding invariant assertions for proving programs. In *ACM SIGPLAN Notices*, volume 10, pages 165–171. ACM, ACM, 1975.

[20] N. Carlini, G. Katz, C. Barrett, and D. L. Dill. Ground-truth adversarial examples. *ArXiv e-prints*, September 2017.

[21] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. Hidden voice commands. In *25th USENIX Security Symposium (USENIX Security 16), Austin, TX*, 2016.

[22] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., 2006.

[23] Jonathan Chang and Stefan Scherer. Learning representations of emotional speech with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1705.02394*, 2017.

[24] Luigi Cherubini. *A Course of Counterpoint and Fugue*, volume 1. R. Cocks, 1837.

[25] Soumith Chintala, Emily Denton, Arjovsky Martin, and Michael Mathieu. How to train a GAN? Tips and tricks to make GANs work. `https://github.com/soumith/ganhacks`.

[26] Edmund M. Clarke and Jeannette M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996.

[27] Loek Cleophas, Gerard Zwaan, and Bruce W. Watson. Constructing factor oracles. In *Proceedings of the 3rd Prague Stringology Conference*, 2003.

[28] Darrell Conklin and Ian H Witten. Multiple viewpoint systems for music prediction. *Journal of New Music Research*, 24(1):51–73, 1995.

[29] David Cope. *Computers and Musical Styles*. Oxford University Press, 1991.

[30] David Cope. *Computer models of musical creativity*. MIT Press Cambridge, 2005.

[31] Balázs Csanád Csáji. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24:48, 2001.

[32] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[33] Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, et al. Lasagne: first release. *Zenodo: Geneva, Switzerland*, 2015.

[34] Alexandre Donzé, Ilge Akkaya, Sanjit A. Seshia, Edward A. Lee, and David Wessel. Real-time Control Improvisation for the SmartJukebox, November 2013.

[35] Alexandre Donzé, Sophie Libkind, Sanjit A. Seshia, and David Wessel. Control Improvisation with application to music. Technical Report UCB/EECS-2013-183, EECS Department, University of California, Berkeley, Nov 2013.

[36] Alexandre Donzé, Rafael Valle, Ilge Akkaya, Sophie Libkind, Sanjit A. Seshia, and David Wessel. Machine improvisation with formal specifications. In *Proceedings of the 40th International Computer Music Conference (ICMC)*, 2014.

[37] Shlomo Dubnov and Gérard Assayag. Universal prediction applied to stylistic music generation. In *Mathematics and music*, pages 147–159. Springer, 2002.

[38] Shlomo Dubnov, Gérard Assayag, Gill Bejerano, and Olivier Lartillot. A system for computer music generation by learning and improvisation in a particular style. *IEEE Computer*, 10(38), 2003.

[39] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[40] The Economist. Fake news: you ain't seen nothing yet, 2017.

[41] The Economist. How "fake news" could get even worse, 2017.

[42] Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 269–286. Springer, 2017.

[43] Dawson Engler, David Yu Chen, Seth Hallem, Andy Chou, and Benjamin Chelf. *Bugs as deviant behavior: A general approach to inferring errors in systems code*, volume 35. ACM, 2001.

[44] Joshua Fineberg. Spectral music. *Contemporary Music Review*, 19(2):1–5, 2000.

[45] Daniel J. Fremont, Alexandre Donzé, Sanjit A. Seshia, and David Wessel. Control Improvisation. *CoRR*, abs/1411.0698, 2014.

[46] Daniel J. Fremont, Alexandre Donzé, Sanjit A. Seshia, and David Wessel. Control Improvisation. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015*, pages 463–474, 2015.

[47] Mark Gabel and Zhendong Su. Javert: fully automatic mining of general temporal properties from dynamic traces. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 339–349. ACM, 2008.

[48] Mark Gabel and Zhendong Su. Symbolic mining of temporal specifications. In *Proceedings of the 30th international conference on Software engineering*, pages 51–60. ACM, 2008.

[49] John Gillick, Kevin Tang, and Robert M. Keller. Learning Jazz grammars. In *Proceedings of the SMC 2009 - 6th Sound and Music Computing Conference*, pages 125–130, 2009.

[50] Jonathan Reuven Gillick. *A Clustering Algorithm for Recombinant Jazz Improvisations*. PhD thesis, 2009.

[51] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[52] E. Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

[53] E. Mark Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.

[54] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[55] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[56] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[57] Alex Graves. Generating sequences with recurrent neural networks. *ArXiv e-prints*, August 2013.

[58] Stefan Grossman, Stephen Calt, and Hal Grossman. *Country Blues Songbook*. Oak, 1973.

[59] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of Wasserstein GANs. *arXiv preprint arXiv:1704.00028*, 2017.

[60] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[61] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[62] James Hendler. Avoiding another AI winter. 2008.

[63] Avinash Hindupur. The GAN zoo. `https://github.com/hindupuravinash/the-gan-zoo`.

[64] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv e-prints*, July 2012.

[65] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[66] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[67] Susmit Jha and Sanjit A Seshia. A theory of formal synthesis via inductive learning. *Acta Informatica*, 54(7):693–726, 2017.

[68] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. *ArXiv e-prints*, February 2017.

[69] Robert M. Keller. *How to Improvise Jazz Melodies*, 2012.

[70] Robert M. Keller and David R. Morrison. A grammatical approach to automatic improvisation. In *Proceedings SMC'07, 4th Sound and Music Computing Conference*, pages 330 – 337, 2007.

[71] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[72] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. *ArXiv e-prints*, June 2017.

[73] S. Krishna Kumar. On weight initialization in deep neural networks. *ArXiv e-prints*, April 2017.

[74] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[75] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[76] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[77] Yann LeCun, Corinna Cortes, and Christopher JC Burges. MNIST handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[78] Yann LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, pages 143–155, 1989.

[79] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint arXiv:1609.04802*, 2016.

[80] George Lewis. Too many notes: Computers, complexity and culture in Voyager. *Leonardo Music Journal*, 10:33–39, 2000.

[81] Wenchao Li. *Specification Mining: New Formalisms, Algorithms and Applications*. PhD thesis, EECS Department, University of California, Berkeley, Mar 2014.

[82] Wenchao Li, Alessandro Forin, and Sanjit A. Seshia. Scalable specification mining for verification and diagnosis. In *Proceedings of the 47th design automation conference*, pages 755–760. ACM, 2010.

[83] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160, 1976.

[84] Jack Long. The real book of Blues. In *The real book of Blues*. Wise, Hal Leonard, 1999.

[85] Yanick Lukic, Carlo Vogt, Oliver Dürr, and Thilo Stadelmann. Speaker identification and clustering using convolutional neural networks. In *Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on*, pages 1–6. IEEE, 2016.

[86] Lyrebird. Lyrebird, 2017.

[87] Keeril Makan. An interview with Edmund Campion. *Computer Music Journal*, 28(4):16–24, 2004.

[88] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley. Least Squares generative adversarial networks. *ArXiv e-prints*, November 2016.

[89] Frank J. Massey Jr. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[90] Kenneth McAlpine, Eduardo Miranda, and Stuart Hoggar. Making music with algorithms: A case-study system. *Computer Music Journal*, 23(2):19–30, 1999.

[91] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in Python. In *Proceedings of the 14th python in science conference*, 2015.

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

[93] Rada Mihalcea, Courtney Corley, Carlo Strapparava, et al. Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*, volume 6, pages 775–780, 2006.

[94] W. Thomas Miller, Paul J. Werbos, and Richard S. Sutton. *Neural networks for control*. MIT press, 1995.

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

[96] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[97] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. *Google Research Blog. Retrieved June*, 20:14, 2015.

[98] Nelson Morgan and Hervé Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *Advances in neural information processing systems*, pages 630–637, 1990.

[99] Jérôme Nika and Marc Chemillier. Improtek: integrating harmonic controls into improvisation in the filiation of OMax. In *International Computer Music Conference Proceedings*, pages 180–187, 2012.

[100] Genevieve B. Orr and Klaus-Robert Müller. *Neural networks: tricks of the trade*. Springer, 2003.

[101] Francois Pachet, Pierre Roy, and Gabriele Barbieri. Finite-length Markov processes with constraints. pages 635–642, 2011.

[102] Jean-Francois Paiement, Samy Bengio, and Douglas Eck. Probabilistic models for melodic prediction. *Artificial Intelligence*, 173(14):1266–1274, 2009.

[103] Santiago Pascual, Antonio Bonafonte, and Joan Serrà. SEGAN: Speech enhancement generative adversarial network. *arXiv preprint arXiv:1703.09452*, 2017.

[104] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544, 2016.

[105] Geoffroy Peeters. A large set of audio features for sound description (similarity and classification) in the CUIDADO project. Technical report, IRCAM, 2004.

[106] Dean A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[107] Kishore Prahallad, Anandaswarup Vadapalli, Naresh Elluru, G. Mantena, B. Pulugundla, P. Bhaskararao, H. A. Murthy, S. King, V. Karaiskos, and A. W. Black. The Blizzard challenge 2013–Indian language task. In *Blizzard Challenge Workshop*, volume 2013, 2013.

[108] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[109] Peter J. Ramadge and W. Murray Wonham. Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, 25(1):206–230, 1987.

[110] Jean-Philippe Rameau. *Traité de l'harmonie réduite à ses principes naturels...* JBC Ballard, 1722.

[111] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[112] Frank Rosenblatt. Principles of neurodynamics. 1962.

[113] David E. Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation. chapter Backpropagation: The Basic Theory, pages 1–34. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995.

[114] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[115] Stuart Russell, Peter Norvig, and Artificial Intelligence. Artificial Intelligence: A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[116] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. *CoRR*, abs/1606.03498, 2016.

[117] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

[118] Sanjit A. Seshia. Combining induction, deduction, and structure for verification and synthesis. *Proceedings of the IEEE*, 103(11):2036–2051, Nov 2015.

[119] Sanjit A Seshia. New frontiers in formal methods: Learning, cyber-physical systems, education, and beyond. *CSI Journal of Computing*, 2(4):R1, 2015.

[120] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Towards verified Artificial Intelligence. *ArXiv e-prints*, June 2016.

[121] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1528–1540. ACM, 2016.

[122] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. *ArXiv e-prints*, December 2016.

[123] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *ArXiv e-prints*, December 2013.

[124] Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

[125] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International Conference on Machine Learning*, pages 843–852, 2015.

[126] Neural Information Processing Systems. Adversarial attacks and defenses, 2017.

[127] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[128] L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. *ArXiv e-prints*, November 2015.

[129] Rafael Valle and Adrian Freed. Symbolic music similarity using neuronal periodicity and dynamic programming. In *Mathematics and Computation in Music*, pages 199–204. Springer, 2015.

[130] 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. *CoRR abs/1609.03499*, 2016.

[131] 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: A fully end-to-end text-to-speech synthesis model. *arXiv preprint arXiv:1703.10135*, 2017.

[132] Ben Wegbreit. The synthesis of loop predicates. *Communications of the ACM*, 17(2):102–113, 1974.

[133] Westley Weimer and George C. Necula. Mining temporal specifications for error detection. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 461–476. Springer, 2005.

[134] Wired. Google's artificial brain is pumping out trippy - and pricey - art., 2017.

[135] Qi Wu, Damien Teney, Peng Wang, Chunhua Shen, Anthony Dick, and Anton van den Hengel. Visual question answering: A survey of methods and datasets. *Computer Vision and Image Understanding*, 2017.

[136] Zhizheng Wu, Nicholas Evans, Tomi Kinnunen, Junichi Yamagishi, Federico Alegre, and Haizhou Li. Spoofing and countermeasures for speaker verification: a survey. *Speech Communication*, 66:130–153, 2015.

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

[138] Jinlin Yang, David Evans, Deepali Bhardwaj, Thirumalesh Bhat, and Manuvir Das. Perracotta: mining temporal API rules from imperfect traces. In *Proceedings of the 28th international conference on Software engineering*, pages 282–291. ACM, 2006.

[139] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. MidiNet: A convolutional generative adversarial network for symbolic-domain music generation using 1d and 2d conditions. *CoRR*, abs/1703.10847, 2017.

[140] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *ArXiv e-prints*, June 2015.

[141] Adham Zeidan, Armin Lehmann, and Ulrich Trick. WebRTC enabled multimedia conferencing and collaboration solution. In *WTC 2014; World Telecommunications Congress 2014; Proceedings of*, pages 1–6. VDE, 2014.

[142] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[143] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *ArXiv e-prints*, November 2016.

[144] Xingxing Zhang and Mirella Lapata. Chinese poetry generation with recurrent neural networks. In *EMNLP*, pages 670–680, 2014.

[145] J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. *ArXiv e-prints*, September 2016.