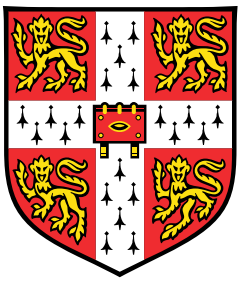
BachBot: A generative model for Bach chorales

Automatic stylistic composition with deep long short-term memory



Feynman Liang

Department of Engineering
University of Cambridge

M.Phil in Machine Learning, Speech, and Language Technology

This dissertation is submitted for the degree of
Masters of Philosophy

Churchill College August 2016

I would like to dedicate this thesis to my loving parents …

Declaration

I, Feynman Liang of Churchill College, being a candidate for the M.Phil in Machine Learning, Speech, and Language Technology, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count:

fliang: Fill this in at the end

Signed:

Date:

Feynman Liang
August 2016

Acknowledgements

And I would like to acknowledge ...

fliang: Make everyone feel appreciated

Mark Gotham : setting research goals, educating me, designing experiments, analyzing neurons, proofreading

Matthew Johnson, Jamie Shotton : proposing project, providing computing resources and regular feedback, promoting online and in MSRC, proofreading

Bill Byrne : setting timeline, general guidance, proofreading

Marcus Tomalin : consultation about music theory and language modelling aspects, designing experiments

Christopher Hicks : providing expert music feedback

Anna Langley : providing user feedback and promoting within CUED

Kyle Kastner : for reviewing a draft of the thesis and providing helpful comments

Tom Nicholson : for reviewing

Niole Nelson : for reviewing

Marcin Tomczak : for discussions

Abstract

This thesis investigates Bach's music composition style using deep sequence learning. We present *BachBot*: a deep LSTM model for automatic stylistic composition of polyphonic music in the style of Bach's chorales. Our approach encodes music scores into a sequential format, reducing the task to one of sequence modeling. Traditional N-gram language models are found to be insufficient, prompting the use of RNN sequence models. We find a 3-layer stacked LSTM performs best and conduct analyses and evaluations to understand its success and failure modes.

Unlike many previous works where model architecture is carefully designed using an understanding of music theory, our methodology intentionally minimizes prior assumptions and utilizes LSTM without modification. While this is not the first application of deep LSTM to Bach chorales, our work consists of the following novel contributions.

First, we devise a sequential encoding for polyphonic music which avoids hand-crafted features and resolves issues noted by prior work. Our improvements include the ability to determine when notes end and a time-resolution at least two times greater than any prior work.

Second, our model analysis uncovers neurons specific to music-theoretic concepts such as chords and cadences. To our knowledge, this is the first reported case demonstrating that LSTM is capable of learning complex harmonic relations automatically from music data.

Finally, we evaluate our automatic composition using a web-based musical Turing test (www.bachbot.com) on a participant pool larger than any previous study on automatic composition (to the best of our knowledge). We demonstrate that high-quality human evaluations, arguably superior to those obtained from Amazon MTurk, can be obtained through a voluntary participation study promoted over social media. The responses recieved found that average participants could differentiate BachBot's generated chorales from bach's originals works only 5% better than random guessing.

UPDATE

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

$\boldsymbol{h}$	hidden state (*i.e.* memory cell contents)
$\boldsymbol{x}$	layer inputs
N_{hid}	dimensionality of hidden state
N_{in}	dimensionality of inputs
N_{out}	dimensionality of outputs
$\boldsymbol{y}$	layer outputs
T	total number of timesteps in a sequence
$\boldsymbol{W}$	weight matrix

Greek Symbols

σ	elementwise activation function
θ	Model Parameters

Superscripts

$\mathscr{E}$	Error or Loss
$\mathscr{E}_t$	Error or Loss at time t
$\boldsymbol{f}_t$	Forget gate values at time t
$\boldsymbol{i}_t$	Input gate values at time t
(l)	layer index in multi-layer networks

o_t Output gate values at time t

Subscripts

st connections from source s to target t

t time index

Other Symbols

$\odot$ Elementwise multiplication

Acronyms / Abbreviations

A Alto

AT Alto and Tenor

ATB Alto, Tenor, and Bass

B Bass

BPTT Backpropagation Through Time

BWV Bach-Werke-Verzeichnis numbering system for Bach chorales

CEC Constant Error Carousel

CPU Central Processing Unit

DAG Directed Acyclic Graph

FER Frame Error Rate

GPU Graphics Processing Unit

LSTM Long Short Term Memory

MIDI Musical Instrument Device Interface

RNN Recurrent Neural Network

SATB Soprano, Alto, Tenor, and Bass

S Soprano

TER Token Error Rate

T Tenor

Since I have always preferred making plans to executing them, I have gravitated towards situations and systems that, once set into operation, could create music with little or no intervention on my part. That is to say, I tend towards the roles of planner and programmer, and then become an audience to the results.

Alpern [3]

1 Introduction

1.1 Motivation

Can the style of a particular composer or genre of music be codified into a deterministic computable algorithm? While it may be easy to enumerate some musical rules, reaching consensus on a formal theory for stylistic composition has proven to be difficult. Even after hundreds of years of study, many modern music theorists would still feel uncomfortable claiming a "correct" algorithm for composing music like Bach, Beethoven, or Mozart.

Despite these difficulties, recent advances in computing and progress in modelling techniques has enabled computational modelling to provide novel insights into various musical phenomena. By offering a method for quantitatively testing theories, computational models can help us learn more about the various cognitive and perceptual processes related to music comprehension, production, and style.

One primary use case for computational music models is **automatic composition**, a task concerned with algorithmic production of musical compositions. While early automatic composition models were predominantly rule-based, the field has experienced an increased interest in connectionist neural-network models over the last 25 years. The recent empirical triumphs of deep learning, a specific form of connectionist modelling, has further fueled the renewed interest in connectionist systems for automatic composition.

1.2 Research aims and scope

This thesis is concerned with **automatic stylistic composition**, where the goal is to create a system capable of generating music in a style similar to a particular composer or genre. We restrict our attention to a particular class of model: **generative probabilistic sequence models** which are **learned from data**. A generative probabilistic model is desirable because it can be applied to a variety of automatic composition tasks, including: harmonizing a melody (by conditioning the model on the melody), automatic composition (by sampling the model), and scoring (by evaluating the model on a given sequence). Fitting the model to data enables it to automatically learn the relationships and regularities present throughout the training data, enabling generation of music which is statistically similar to what was observed during training.

We develop a method for automatic stylistic composition which brings together ideas from deep learning, language modelling, and music theory. Our motivation stems from recent developments [120, 77, 45, 116] which have enabled deep learning models to surpass prior state-of-the-art techniques in domains such as computer vision, natural language processing, and speech recognition. As it has already shown promise across a wide variety of problem domains, we hypothesized that the application of modern deep learning techniques to automatic composition would yield similar success.

The aim of our research is **to build an automatic composition system capable of imitating Bach's composition style on both harmonization and automatic composition tasks in a manner that an average listener finds indistinguishable from Bach**. While the method we develop is capable of modelling arbitrary polyphonic music compositions, we restrict the scope of our study to Bach's chorales. These provide a relatively large corpus by a single composer, are well understood by music theorists, and are routinely used when teaching music theory.

1.3 Organization of the chapters

The remaining chapters are organized as follows:

Chapter 4 on page 45 describes the construction and evaluation of our final model. Our approach first encodes music scores into a sequential format. reducing the task to one of sequence modelling. This type of problem is analagous to that of language modelling in speech research. Unfortuantely, we found that traditional N-gram models performed poorly because they are unable to capture the important long-range dependencies and precise harmonic rules present in music. Inspired by the strong performance of recurrent neural network language models, we then investigated sequence models parameterized by recurrent neural networks and found that a deep long short-term memory architecture performs particularly well.

In **??** on page ??, we open the black box and characterize the internals of our learned model. Through measuring neuron activations to applied stimulus, we discover that the certain neurons in the model have specialized to specific musical concepts without any form of supervision or prior knowledge. Our results here represent a significant milestone in computational modelling of how musical knowledge is acquired.

We turn to the task of harmonization in chapter 6 on page 67 and present a method for conditionally sampling our model in order to generate harmonizations.

To evaluate our success in achieving our stated research aim, chapter 7 on page 73 describes the design, results, and conclusions from a large-scale musical Turing test we conducted. Encouragingly, we find that average participants are only 5% more likely than random chance to differentiate BachBot from real Bach. Furthermore, our analysis of participant demographics and costs suggest that voluntary participation user studies promoted over social media yields superior data than paid studies conducted using Amazon MTurk. This finding is especially significant to the field of machine translation, where use of MTurk in academic publications is widely accepted.

Finally, we summarize the conclusions from our work and suggest future directions for extension in chapter 8 on page 85.

2 Background

The goal of this chapter is to provide only the necessary background in recurrent neural networks and generative probabilistic sequence modelling required for understanding our models, experiments, and results. It also introduces some common definitions and clarifies notation used throughout later chapters.

A basic understanding of Western music theory and neural networks is assumed. Readers unfamiliar with concepts such as piano rolls, Roman numeral analysis, and cadences, should review chapter A on page 99 for a quick primer and Piston [107] and Denny [36] for more thorough coverage. Likewise, those whom wish to review concepts such as activation functions, neurons, and applying recurrent neural networks over arbitrary length sequences are advised to review chapter B on page 107 and consult Bengio [11] for further reference.

2.1 Recurrent neural networks

2.1.1 Notation

We begin by clarifying common notation and conventions used to describe multi-layer **recurrent neural networks** (RNNs). Unless otherwise specified, future use of notation should be interpreted as defined in this section.

We use the subscript $t \in \{1, 2, \cdots, T\}$ to denote the **time index** within a sequence of length $T \in \mathbb{N}$

A sequence of **inputs** is denoted by $\boldsymbol{x}$ and the sequence elements at timestep t is denoted by $\boldsymbol{x}_t \in \mathbb{R}^{N_{in}}$ and assumed to have dimensionality $N_{in} \in \mathbb{N}$. Similarly, $\boldsymbol{h}_t \in \mathbb{R}^{N_{hid}}$ and $\boldsymbol{y}_t \in \mathbb{R}^{N_{out}}$ denote elements from the **hidden state** and **output** sequences respectively.

To describe model parameters, we use $\boldsymbol{W}$ to indicate a real-valued **weight matrix** consisting of all the connection weights between two sets of neurons and $\sigma(\cdot)$ to indicate an element-wise **activation function**. The collection of all model parameters is denoted by $\boldsymbol{\theta}$.

When further clarity is required, we use subscripts $\boldsymbol{W}_{st}$ denote the connection weights from a set of neurons s to another set of neurons t (*i.e.* in section 2.1.5 on page 32, $\boldsymbol{W}_{xf}$ and $\boldsymbol{W}_{xh}$ refer to the connections from the inputs to the forget gate and hidden state respectively). Subscripts on activation functions $\sigma_{s,t}(\cdot)$ are to be interpreted analogously.

Equipped with the above notation, the equations for **RNN time dynamics** (review chapter B on page 107 if this is unfamiliar) can be expressed as

$$\left.\begin{aligned} \boldsymbol{h}_t &= \boldsymbol{W}_{xh}\sigma_{xh}\left(\boldsymbol{x}_t\right) + \boldsymbol{W}_{hh}\sigma_{hh}\left(\boldsymbol{h}_{t-1}\right) \\ \boldsymbol{y}_t &= \boldsymbol{W}_{hy}\sigma_{hy}\left(\boldsymbol{h}_t\right) \end{aligned}\right\} \quad \text{RNN time dynamics} \tag{2.1}$$

When discussing multi-layer networks, we use $L \in \mathbb{N}$ to denote total number of layers and parenthesized superscripts (l) for $l \in \{1, 2, \cdots, L\}$ to indicate the layer. For example, $\boldsymbol{z}_t^{(2)}$ is the hidden states of the second layer and $N_{in}^{(3)}$ is the dimensionality of the third layer's inputs $\boldsymbol{x}_t^{(3)}$. Unless stated otherwise, multi-layer networks will assume that the outputs of the $l-1$st layer are used as the inputs of the lth layer (*i.e.* $\forall t : \boldsymbol{x}_t^{(l)} = \boldsymbol{y}_t^{(l-1)}$).

2.1.2 The memory cell abstraction

While a large number of proposed RNN variants exist [46, 79, 72, 20, 83, 94], most share the same underlying structure and differ only in small localized implementation details. Encapsulating these differences and abstracting them away enables general discussion about RNN architecture without making a specific choice on implementation.

To do so, we introduce the **memory cell** abstraction to encapsulate the details of computing $\boldsymbol{y}_t$ and $\boldsymbol{h}_t$ from $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$. This is illustrated visually in fig. 2.1, which shows a standard Elman-type RNN [46] with the memory cell indicated by a dashed box isolating the recurrent hidden state. The edges entering the memory cell ($\boldsymbol{x}_t$, $\boldsymbol{h}_{t-1}$) make up the **memory cell inputs** and the outgoing edges ($\boldsymbol{y}_t$, $\boldsymbol{h}_t$) are the **memory cell outputs**, so a concrete implementation of a memory cell needs to provide two functions $\boldsymbol{f}_h$ and $\boldsymbol{f}_y$ which uses $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$ to compute the next hidden state $\boldsymbol{h}_t = f_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$ and output $\boldsymbol{y}_t = f_y(\boldsymbol{h}_t)$.

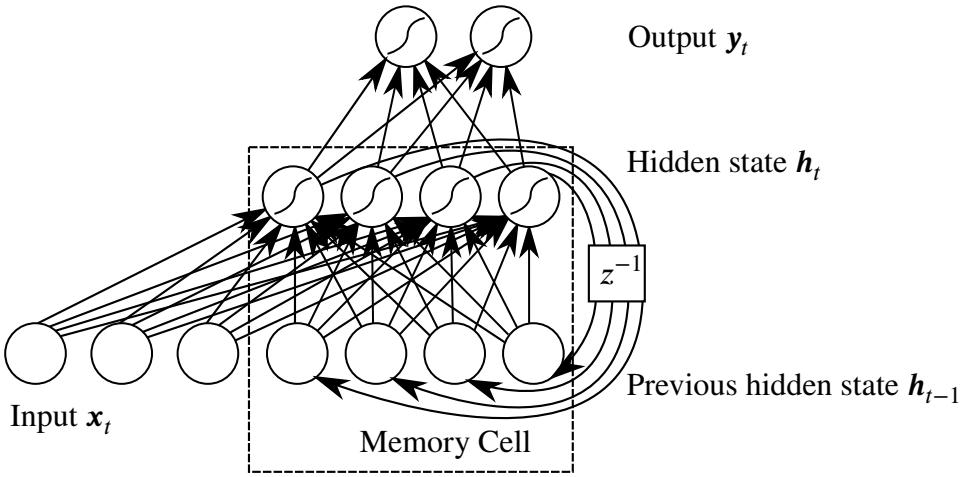


Fig. 2.1 An Elman-type RNN with a single hidden layer. The recurrent hidden state is illustrated as unit-delayed (denoted by z^{-1}) feedback edges from the hidden states to the input layer. The memory cell encapsulating the hidden state is also shown.

2.1.3 Operations on RNNs: stacking and unrolling

Stacking memory cells to form deep RNNs

Just like deep neural networks, RNNs can be **stacked** to form deep RNNs [45, 116] by treating the outputs from the $l-1$st layer's memory cells as inputs to the lth layer (see fig. 2.2).

Prior work has observed that "deep RNNs outperformed the conventional, shallow RNN" Pascanu et al. [104], affirming the importance of stacking multiple layers in RNNs. The improved modelling expressivity can be attributed to two primary factors: composition of multiple non-linear activation functions and an increase in the number of paths for backpropagated error signals to flow. The former reason is analogous to the case in deep belief networks, which is well documented [11]. To understand the latter, notice that in fig. 2.2 there is only a single path from $\boldsymbol{x}_{t-1}$ to $\boldsymbol{y}_t$ hence the conditional independence $\boldsymbol{y}_t \perp\!\!\!\perp \boldsymbol{x}_{t-1} | \boldsymbol{h}_t^{(1)}$ is satisfied. However, in fig. 2.2 there are multiple paths from $\boldsymbol{x}_{t-1}$ to $\boldsymbol{y}_t$ (*e.g.* passing through either $\boldsymbol{h}_{t-1}^{(2)} \rightarrow \boldsymbol{h}_t^{(2)}$ or $\boldsymbol{h}_{t-1}^{(1)} \rightarrow \boldsymbol{h}_t^{(1)}$) through which information may flow.

Unrolling RNNs into directed acyclic graphs

Given an input sequence $\{\boldsymbol{x}\}_{t=1}^{T}$, an RNN can be **unrolled** into a **directed acyclic graph** (DAG) comprised of T copies of the memory cell connected forwards in time. This is illustrated for a stacked 2-layer RNN in Figure 2.2, where the vectors $\boldsymbol{y}_t$, $\boldsymbol{h}_t$, and $\boldsymbol{x}_t$ are depicted as blocks and the $\boldsymbol{h}_t$ is understood to represent a memory cell.

Figure 2.2 shows that the hidden state $\boldsymbol{h}_t$ is passed forwards throughout the sequence of computations. This gives rise to an alternative interpretation of the hidden state as a temporal

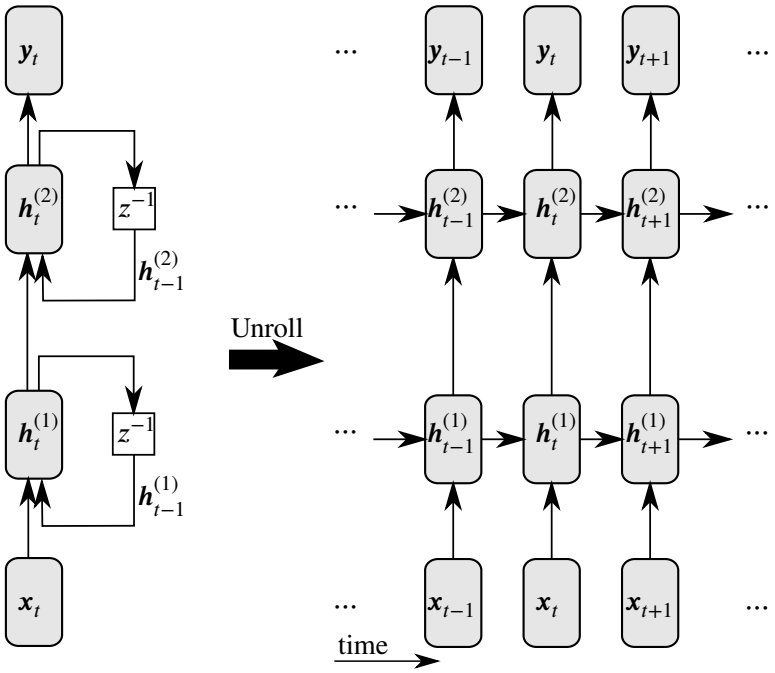


Fig. 2.2 Block diagram representation of a -layer RNN (left) and its corresponding DAG (right) after unrolling. The blocks labelled with $\boldsymbol{h}_t$ represent memory cells whose parameters are shared across all times t.

memory mechanism. Under this interpretation, updating the hidden state $\boldsymbol{h}_t = f_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$ can be viewed as **writing** information from the current inputs $\boldsymbol{x}_t$ to memory and producing the outputs $\boldsymbol{y}_t = f_y(\boldsymbol{h}_t)$ can be interpreted as **reading** information from memory.

2.1.4 Training RNNs and backpropagation through time

The parameters $\boldsymbol{\theta}$ of a RNN are typically learned from data by minimizing some **cost** $\mathscr{E} = \sum_{1 \leq t \leq T} \mathscr{E}_t(\boldsymbol{x}_t)$ measuring the performance of the network on some task. This optimization is usually performed using iterative methods which require computation of gradients $\frac{\partial \mathscr{E}}{\partial \boldsymbol{\theta}}$ at each iteration.

In feed-forward networks, computation of gradients can be performed efficiently using backpropagation [18, 88, 114]. While time-delayed recurrent hidden state connections appear to complicate matters initially, unrolling the RNN removes the time-delayed recurrent edges and converts the RNN into a DAG (*e.g.* fig. 2.2) which can be interpreted as a T layered feed-forward neural network with parameters shared across all T layers.

This view of unrolled RNNs as feedforward networks motivates **backpropagation through time** (BPTT) [60], a method for training RNNs which applies backpropagation to the unrolled DAG. Applying the chain rule to the RNN dynamics equations (eq. (2.1) on page 28) unrolled

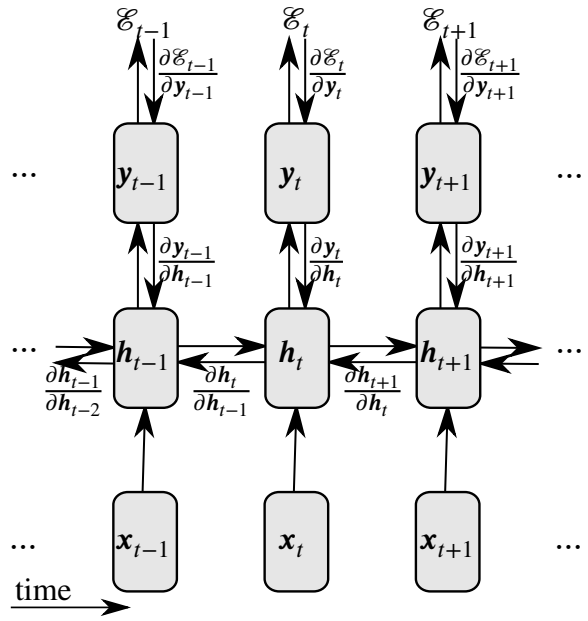


Fig. 2.3 The gradients passed along network edges during BPTT.

network (see fig. 2.3), we obtain

$$\frac{\partial \mathscr{E}}{\partial \boldsymbol{\theta}} = \sum_{1 \le t \le T} \frac{\partial \mathscr{E}_t}{\partial \boldsymbol{\theta}} \tag{2.2}$$

$$\frac{\partial \mathscr{E}_t}{\partial \boldsymbol{\theta}} = \sum_{1 \le k \le t} \left(\frac{\partial \mathscr{E}_t}{\partial \boldsymbol{y}_t} \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{\theta}} \right) \tag{2.3}$$

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} = \prod_{t \ge i > k} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} = \prod_{t \ge i > k} \boldsymbol{W}_{hh}^{\intercal} \operatorname{diag}\left(\sigma'_{hh}(\boldsymbol{h}_{i-1})\right) \tag{2.4}$$

Equation (2.3) expresses how the error $\mathscr{E}_t$ at time t is a sum of **temporal contributions** $\frac{\partial \mathscr{E}_t}{\partial \boldsymbol{y}_t} \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{\theta}}$ measuring how $\boldsymbol{\theta}$'s impact on $\boldsymbol{h}_k$ affects the cost $\mathscr{E}_t$ at some future time $t > k$. The quantity $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k}$ in eq. (2.4) measures the affect of the hidden state $\boldsymbol{h}_k$ on some future state $\boldsymbol{h}_t$ where $t > k$ and can be interpreted as transferring the error "in time" from step t back to step k [103].

Just like traditional backpropagation, fig. 2.3 demonstrates how BPTT divides the computation of a global gradient $\frac{\partial \mathscr{E}}{\partial \theta}$ into a series of local gradient computations, each of which involves significantly less variables and hence is easier to compute.

Vanishing/exploding gradients

Naive implementations of RNNs (specifically eq. (2.1)) often suffer from two well known problems: the **vanishing gradient** and **exploding gradient** [13]. These problems are both related to the product in eq. (2.4) exponentially growing or shrinking over long time-spans (*i.e.* $t \gg k$). A sufficient condition (proved in **??** on page ??) for vanishing gradients is

$$\left\|\boldsymbol{W}_{hh}\right\| < \frac{1}{\gamma_\sigma} \tag{2.5}$$

where $\|\cdot\|$ is the matrix operator norm (see **??** on page ??), $\boldsymbol{W}_{hh}$ is defined in eq. (2.1) on page 28, and γ_σ is a constant depending on the choice of activation function (*e.g.* $\gamma_\sigma = 1$ for $\sigma_{hh} = \tanh$, $\gamma_\sigma = 0.25$ for $\sigma_{hh} = \text{sigmoid}$).

This difficulty learning relationships between events spaced far apart in time presents a significant challenge for music applications. As noted by Cooper and Meyer [25]:

> Long-term dependencies are at the heart of what defines a style of music, with events spanning several notes or bars contributing to the formation of metrical and phrasal structure.

2.1.5 Long short term memory: solving the vanishing gradient

In order to build a model which learns long range dependencies, vanishing gradients must be avoided. A popular memory cell architecture which does so is **long short term memory** (LSTM). Proposed by Hochreiter and Schmidhuber [72], LSTM solves the vanishing gradient problem by enforcing **constant error flow** on eq. (2.4), that is

$$\boldsymbol{W}_{hh}^{\intercal}\sigma'_{hh}(\boldsymbol{h}_t) = \boldsymbol{I} \tag{2.6}$$

where $\boldsymbol{I}$ is the identity matrix.

As a result of the constant error flow condition, notice that eq. (2.4) on page 31 becomes

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} = \prod_{t \geq i > k} \boldsymbol{W}_{hh}^{\intercal} \operatorname{diag}\left(\sigma'_{hh}(\boldsymbol{h}_{i-1})\right) = \prod_{t \geq i > k} \boldsymbol{I} = \boldsymbol{I} \tag{2.7}$$

The dependence on the time-interval $t - k$ is no longer present, ameliorating the exponential decay causing vanishing gradients and enabling long-range dependencies (*i.e.* $t \gg k$) to be learned.

Integrating eq. (2.6) yields $\boldsymbol{W}_{hh}\sigma_{hh}(\boldsymbol{h}_t) = \boldsymbol{h}_t$. Since this must hold for any hidden state $\boldsymbol{h}_t$, this means that:

1. $\boldsymbol{W}_{hh}$ must be full rank
2. σ_{hh} must be linear
3. $\boldsymbol{W}_{hh}\sigma_{hh} = \boldsymbol{I}$

In the **constant error carousel** (CEC), this is ensured by setting $\sigma_{hh} = \boldsymbol{W}_{hh} = \boldsymbol{I}$. This may be interpreted as removing time dynamics on $\boldsymbol{h}$ in order to permit error signals to be transferred backwards in time (eq. (2.4)) without modification (*i.e.* $\forall t \geq k : \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} = \boldsymbol{I}$).

In addition to using a CEC, a LSTM introduces three gates controlling access to the CEC:

Input gate : scales input $\boldsymbol{x}_t$ elementwise by $\boldsymbol{i}_t \in [0, 1]$, **writes** to $\boldsymbol{h}_t$

Output gate : scales output $\boldsymbol{y}_t$ elementwise by $\boldsymbol{o}_t \in [0, 1]$, **reads** from $\boldsymbol{h}_t$

Forget gate : scales previous cell value $\boldsymbol{h}_{t-1}$ by $\boldsymbol{f}_t \in [0, 1]$, **resets** $\boldsymbol{h}_t$

Mathematically, the LSTM model is defined by the following set of equations:

$$\boldsymbol{i}_t = \text{sigmoid}(\boldsymbol{W}_{xi}\boldsymbol{x}_t + \boldsymbol{W}_{yi}\boldsymbol{y}_{t-1} + \boldsymbol{b}_i) \tag{2.8}$$
$$\boldsymbol{o}_t = \text{sigmoid}(\boldsymbol{W}_{xo}\boldsymbol{x}_t + \boldsymbol{W}_{yo}\boldsymbol{y}_{t-1} + \boldsymbol{b}_o) \tag{2.9}$$
$$\boldsymbol{f}_t = \text{sigmoid}(\boldsymbol{W}_{xf}\boldsymbol{x}_t + \boldsymbol{W}_{yf}\boldsymbol{y}_{t-1} + \boldsymbol{b}_f) \tag{2.10}$$
$$\boldsymbol{h}_t = \boldsymbol{f}_t \odot \boldsymbol{h}_{t-1} + \boldsymbol{i}_t \odot \tanh(\boldsymbol{W}_{xh}\boldsymbol{x}_t + y_{t-1}\boldsymbol{W}_{yh} + \boldsymbol{b}_h) \tag{2.11}$$
$$\boldsymbol{y}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{h}_t) \tag{2.12}$$

where $\odot$ denotes elementwise multiplication of vectors.

Notice that the gates ($\boldsymbol{i}_t$, $\boldsymbol{o}_t$, and $\boldsymbol{f}_t$) controlling flow in and out of the CEC are all time varying. This can be interpreted as a mechanism enabling LSTM to explicitly learn which error signals to trap in the CEC and when to release them [72], enabling error signals to potentially be transported across long time lags.

Some authors define LSTM such that $\boldsymbol{h}_t$ is not used to compute gate activations, referring to the case where $\boldsymbol{h}_t$ is connected as "peephole connections" [55]. We will use LSTM to refer to the system of equations as written above.

Practicalities for successful applications of LSTM

Many applications of LSTM [38, 143, 104] share some common practical techniques for ensuring successful training. Perhaps most important is **gradient norm clipping** [93, 103] where the gradient is scaled or clipped whenever it exceeds a threshold. This is necessary because while

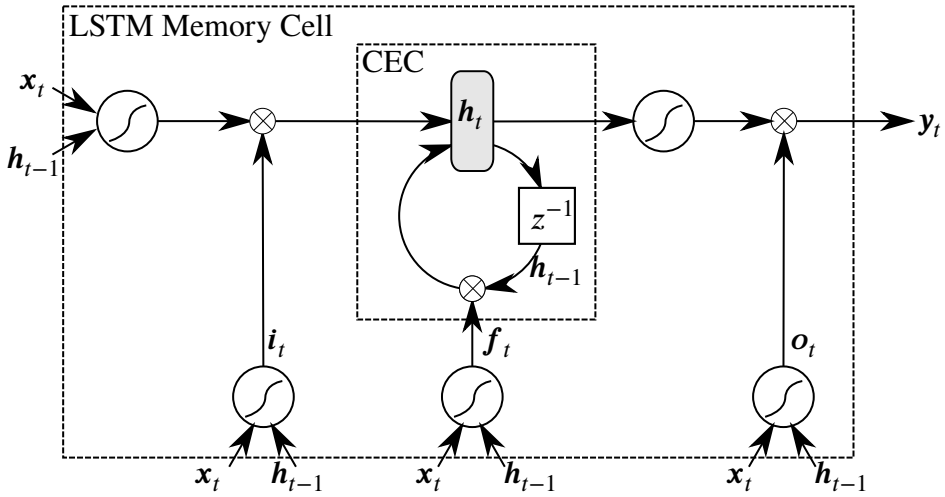


Fig. 2.4 Schematic for a single LSTM memory cell. Notice how the gates $\boldsymbol{i}_t$, $\boldsymbol{o}_t$, and $\boldsymbol{f}_t$ control access to the constant error carousel (CEC).

vanishing gradients are mitigated by CECs, LSTM do not explicitly protect against exploding gradients.

Another common practice is the use of methods for reducing overfitting and improving generalization. In particular, **dropout** [120] can be applied to the connections between memory cells in a stacked RNN to regularize the learned features to be more robust to noise [144]. Additionally, **batch normalization** [77] can also be applied to to the memory cell hidden states to reduce co-variate shifts, accelerate training, and improve generalization.

Finally, applications of RNNs to long sequences can incur a prohibitively high cost for a single parameter update [125]. For instance, computing the gradient of an RNN on a sequence of length 1000 costs the equivalent of a forward and backward pass on a 1000 layer feed-forward network. This issue is typically addressed by only backpropagating error signals a fixed number of timesteps back in the unrolled network, a technique known as **truncated BPTT** [138]. As the hidden states in the unrolled network have nevertheless been exposed to many timesteps, learning of long range structure is still possible.

2.2 Sequence probability modelling

fliang: Introduce this stuff better (should we assume LSTM after this point)

In order to use LSTM as a model for music, the following assumptions about the sequences $\boldsymbol{x}_{1:T}$, $\boldsymbol{y}_{1:T}$, and $\boldsymbol{h}_{0:T}$ are made:

1. Modified Markov assumption:

$$\forall t \,:\, P(\boldsymbol{h}_t|\boldsymbol{h}_{0:t-1}, \boldsymbol{x}_{1:t}) = P(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) \tag{2.13}$$

2. Hidden State Stationarity:

$$\forall t_1, t_2 \,:\, P(\boldsymbol{h}_{t_1} = \boldsymbol{k}|\boldsymbol{h}_{t_1-1} = \boldsymbol{i}, \boldsymbol{x}_{t_1} = \boldsymbol{j}) = P(\boldsymbol{h}_{t_2} = \boldsymbol{k}|\boldsymbol{h}_{t_2-1} = \boldsymbol{i}, \boldsymbol{x}_{t_2} = \boldsymbol{j}) \tag{2.14}$$

3. Output Stationarity:

$$\forall t_1, t_2 \,:\, P(\boldsymbol{y}_{t_1} = \boldsymbol{j}|\boldsymbol{h}_{t_1} = \boldsymbol{i}) = P(\boldsymbol{y}_{t_2} = \boldsymbol{j}|\boldsymbol{h}_{t_2} = \boldsymbol{i}) \tag{2.15}$$

4. Output independence:

$$P(\boldsymbol{y}_{1:T}|\boldsymbol{h}_{0:T}, \boldsymbol{x}_{1:T}) = \prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t, \boldsymbol{x}_t) \tag{2.16}$$

These assumptions imply the sequential factorization:

$$\begin{aligned}
&P(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{h}_0, \boldsymbol{x}_{1:T}) && && (2.17)\\
&= P(\boldsymbol{y}_{1:T}|\boldsymbol{h}_{0:T}, \boldsymbol{x}_{1:T}) P(\boldsymbol{h}_{1:T}|\boldsymbol{h}_0, \boldsymbol{x}_{1:T}) && && (2.18)\\
&= \left(\prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t)\right) P(\boldsymbol{h}_{1:T}|\boldsymbol{h}_0, \boldsymbol{x}_{1:T}) && \text{eq. (2.16)} && (2.19)\\
&= \left(\prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t)\right)\left(\prod_{t=1}^{T} P(\boldsymbol{h}_t|\boldsymbol{h}_{0:t-1}, \boldsymbol{x}_{1:t})\right) && && (2.20)\\
&= \left(\prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t)\right)\left(\prod_{t=1}^{T} P(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)\right) && \text{eq. (2.13)} && (2.21)\\
&= \prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t, \boldsymbol{x}_t) P(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) && && (2.22)\\
& && && (2.23)
\end{aligned}$$

Together, eq. (2.14) and eq. (2.15) imply that $P(\boldsymbol{y}_t|\boldsymbol{h}_t)$ and $P(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$ are time-invariant and can be modelled by the same recurrent function.

In RNNs, the hidden state dynamics $P(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$ are deterministic:

$$\boldsymbol{h}_t = f_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}) \tag{2.24}$$

Which means that $P(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{h}_0, \boldsymbol{x}_{1:T}) = P(\boldsymbol{y}_{1:T}|\boldsymbol{h}_0, \boldsymbol{x}_{1:T})$. This yields the factorization

$$P(\boldsymbol{y}_{1:T}|\boldsymbol{h}_0, \boldsymbol{x}_{1:T}) = P(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{h}_0, \boldsymbol{x}_{1:T}) = \prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t, \boldsymbol{x}_t) f_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}) \tag{2.25}$$

fliang: Draw PGM

However, one minor problem remains. Let $\boldsymbol{z}_t = f_y(f_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}))$ (with f_y and f_h as defined in

fliang: ref

) denote the outputs of the RNN model at time t. Note that $\boldsymbol{z}_t$ can be any real vector in $\mathbb{R}^{|V|}$

fliang: Define V to be the vocabulary

, but $P(\boldsymbol{x}_{t+1}|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$ is a probability vector constrained to sum to one.

Fortunately, we can treat $\boldsymbol{z}_t$ as the **scores** for a **Boltzmann distribution**

$$P(\boldsymbol{y}_t = s|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) = \frac{\exp\left(-\boldsymbol{z}_{t,s}/T\right)}{\sum_{k=1}^{K}\left(\exp -\boldsymbol{z}_{t,k}/T\right)} \tag{2.26}$$

where $T \in \mathbb{R}^+$ is a **temperature** parameter (set to $T = 1$ during training and varied during sampling). To keep notation compact, we omit writing this explicitly and understand $P(\boldsymbol{y}_t|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$ to mean the Boltzmann distribution parameterized by the scores $f_y(f_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}))$.

Note the similarity between eq. (2.13)–eq. (2.16) and the assumptions for Hidden Markov models [110]. Discrepancies are due to the presence of an input sequence $\boldsymbol{x}_{1:T}$ in our sequence-to-sequence model.

fliang: Discuss validity of assumptions, namely output independence assuming hidden state and input summarize all prior context

3 Related Work

fliang: Compare against language modelling research because it's related

fliang: Compare to N-grams; show how it's like an infinite context. One interpretation is to view the hidden state $\boldsymbol{h}_t$ as an infinite-length prior context window, summarizing all of the prior inputs into into a compact fixed-size vector.

Some common trends throughout related works include

1. Use of domain-specific representations for musical data
2. Modelling at multiple resolutions / timescales (*i.e.* chords vs notes)

In contrast, we avoid imposing prior knowledge in order to avoid any biases and hope that the model will learn the features relevant for good performance.

3.1 Machine learning on musical data

Computational methods applied to large corpora of music was first described in Coutinho et al. [30], which termed the phrase "computational musicology." Since then, development modern tools have greatly aided research efforts. `music21` [32] is a Python programming environment for performing computations over musical composition data which has been utilized for a variety of computational musicology tasks ranging from hierarchical modelling of metrical

structure [6], feature generation for downstream machine learning[33], and style classification [69].

Focusing on machine learning applications, most research can be classified under one or more of the following tasks:

1. Classification: the style, composer, or other musical attribute is to be classified
2. Harmonization: a melody is given and the remaining parts are to be generated
3. Completion: given the beginning of a score, the remainder is to be generated
4. Automatic Composition: a complete unconstrained score is to be generated

There is a vast body of research dealing with music classification tasks, including: style classification [69, 35], automated harmonic analysis [100], information retrieval [92], and performer identification [121]. However, it is not straightforward to utilize work in this area to solve our research goals of music synthesis.

fliang: of what?

.

Research interests within automatic composition ranges includes generating melody lines [24, 129, 130], producing harmonizations or accompaniments [41, 70, 132, 4, 2, 21], and generating full-length novel compositions [46, 119, 44, 117].

fliang: emphasize stylistic part

In addition, our work primarily on **automatic stylistic composition** where we impose the additional constraint that the generated music should be stylistically similar to a particular genre or artist.

3.2 Models for automatic composition

Unlike classification, the other tasks (harmonization, completion, automatic composition) require synthesis of novel music. In a review by Toiviainen [131], automatic composition methods are broadly classified as either symbolic (*i.e.* rule based) or connectionist (*i.e.* neural networks).

3.2.1 Symbolic rule-based methods

Symbolic methods are popular due to their high degree of interpretability. As described by Todd [130], symbolic methods enable composers to write down the composition rules employed in their own creative process and then use a computer to execute these instructions,

enabling assessment of whether the results of the rules held artistic merit. This approach has been prevalent in automatic composition since the 1960s [130].

CHORAL [41] is one of the first rule based expert system for harmonising Bach chorales. It uses 350 manually defined rules as well as hand-tuned search heuristics.

Kulitta [108] is a recent rule-based system based on Schenkerian harmonic theory[115]. It extends learning algorithms for probabilistic context free grammars (PCFGs) to learn a "probabilistic temporal graph grammar" [109] which is then used for stylistic composition. They evaluate on Amazon MTurk with 237 participants using a Likert scale rating system [87] and found their system to be more similar to Bach than a random walk.

Experiments in Music Intelligence (EMI) [27, 26] proposed a system which automatically extracted rules to build an augmented transition network[135]. The system was capable of reproducing music to a particular genre or author, suggesting that the rules extracted by the system can capture a sense of musical style.

fliang: stylistic composition Hörnel [74], Cruz-Alcázar and Vidal-Ruiz [31], Eck and Schmidhuber [42], Chuan and Chew [21], Sturm, Santos, and Korshunova [124], and Collins et al. [22]

Genetic algorithms operating on symbolic encodings of music scores have also received significant attention Weinberg et al. [136] and Cope [29]. Emmy and Emily Howell [28] are extension of EMI, which uses EMI as a database of compositions to recombine and build novel compositions from [29].

In Cruz-Alcázar and Vidal-Ruiz [31], grammatical inference is used to learn regular grammars over chord progressions for modelling musical style. Tsang and Aitken [132] applies constraint logic programming for generating harmonizations which satisfy certain harmonic constraints.

While symbolic methods can easily incorporate domain-specific knowledge and are more interpretable than connectionist models, they are also inherently biased by their creators' subjective theories on harmony and music cognition. Furthermore, specification of hand-crafted rules is a laborious process which requires significant music experience and does not improve when given larger amounts of data. Additionally, rule-based methods are brittle to distortion and noise. Furthermore, they limit creativity by disallowing deviation from the defined rules.

3.2.2 Connectionist methods

Connectionism, also known as parallel distributed processing, is performed by a collection of several simple processing units connected in a network and acting in cooperation [66]. This shift in paradigm replaces strict rule-following behaviour with regularity-learning and generalization [39].

Neural networks have been previously applied to music with varying degrees of success[65]. The earliest connectionist music models utilized note-level Jordan RNNs on melody generation and harmonization tasks [129, 130, 14].

fliang: say more here

A landmark connectionist system is Mozer's CONCERT [96], a BPTT RNN for note-by-note composition. CONCERT models music at two levels of resolution: notes and chords. Notes utilize a psychologically-based representation [118] and chords use a distributed embedding originally trained for style classification [85]. The model passes objective evaluations by faithfully reproducing scales but "while the local contours made sense, the pieces were not musically coherent, lacking thematic structure and having minimal phrase structure and rhythmic organisation" (Mozer [96]).

Boulanger-Lewandowski, Vincent, and Bengio [15] proposed the RNN-RBM. a time-varying RBM with hidden units evolving over time according to a RNN, to model polyphonic music on a piano roll representation. However, training the RNN-RBM requires an expensive contrastive divergence sampling step at each timestep and a nontrivial Hessian-free optimisation routine. Furthermore, the authors quantized music to eighth-notes. In contrast, our work uses the well-understood truncated BPTT algorithm for training and quantizes to sixteenth-notes to achieve two-times higher time resolution.

Lyu [91] extended the RNN-RBM[15] to use a LSTM instead of a RNN for modelling hidden unit time dynamics. Unfortunately, it suffers from many of the same problems such as lack of meaningful evaluation.

fliang: We should have a subsection on evaluation of automatic composition systems

fliang: Goel, Vohra, and Sahoo [59] tackles polyphonic modeling with RNN-DBN

3.2.3 Hybrid methods

Hybrid approaches which combine both rule-based and connectionist methods have also been investigated. One of the first hybrid systems for music synthesis is HARMONET [70], which combines connectionist neural networks with formal rules to specifically harmonize Bach chorales. It implements a domain-specific processing pipeline consisting of:

1. Harmonic modelling: Predict harmonic skeleton (*i.e.* Roman numerals quantized to quarter-notes) using a neural network
2. Expand each Roman numeral to chords using formal rules
3. Ornamentation: add eighth-notes using formal rules

The specialized architecture of HARMONET makes it unable to generalize to other tasks such as automatic composition or composition scoring

fliang: Define this task

. Additionally, the use of formal rules makes the system suffer from the same problems that rule-based systems suffer from.

MELONET [47] builds on top of HARMONET's harmonic modelling to construct chorale partitas (*i.e.* variations where one of the parts is varied in a harmonically believable way). It first introduced multiple ideas which have been rediscovered in recent years, including:

1. Delayed update units to model multiple timescales (described again in Clockwork RNNs [83])

2. Use of Resilient Propagation (RProp) [112] for training (described again in Liu and Ramakrishnan [89])

Additionally, MELONET utilizes a motif classification neural network to explicitly force motifs to appear multiple times within a partita. Follow up work by Hornel and Ragg [75] extends MELONET to use a distributed representation for motifs and a genetic algorithm for training. While MELONET introduces many novel ideas, its limited training set size (16 Pachelbel chorales [74]) and domain-specific architecture limit the generalizability of results.

fliang: Better analyze this

CHIME [48] adopted the Jordan RNN from Todd [130] to add a second training phase using actor-critic reinforcement learning [127]. The critic is constructed using a collection of "music rules," enabling incorporation of prior knowledge.

Eck and Lapalme [43] extends the connectionist model from Eck and Schmidhuber [44] to explicitly account for meter using an autocorrelation-based predictor.

3.2.4 LSTM music synthesis models

Prior work has demonstrated LSTM possesses many properties desirable for music applications. Their superiority over traditional RNNs has been well documented[54]. They can learn to count and measure time intervals between events spaced arbitrarily far apart in time [55], a property N-gram language models do not possess. Gers, Schraudolph, and Schmidhuber [57] demonstrated LSTM learning to produce self-sustaining oscillations at a regular frequency, suggesting that they are capable of discovering periodic structure. Franklin [52] evaluates various RNN architectures on variety of music tasks and concludes: "while we have found a task that challenges a single LSTM network, we do not believe that any other recurrent networks we have used would be able to learn these songs."'

One of the first applications of LSTM to music was by Eck and Schmidhuber [44] and Eck and Schmidhuber [42]. Using a LSTM to model blues chord progressions and another to model melody lines given chords, the authors reported that LSTM can learn long term music structure such as repeated motifs can be learned without explicit modelling (*e.g.* MELONET). However, the music representation quantized to eighth notes, used considered pitch classes without accounting for octaves, and limited the model to 12 possible chords. Additionally, there was "no explicit way to determine when a note ends," prohibiting discrimination between four consecutive articulations of a note at the same pitch from a single note held for four timesteps. In contrast, our model accounts for the octaves in addition to pitch class, does not restrict the possible chords, operates at twice the time resolution, and also models when a note ends.

More recently, Sturm, Santos, and Korshunova [124] and Sturm et al. [123] trained character-level LSTM on 23,000 folk music scores represented in ABC notation[1], a high-level text format for music. ABC format is unsatisfactory for our use case because polyphonic scores are encoded one part at a time so notes sounding close together in time may appear very far apart in the sequence. As a result, it is unsurprising that the authors do not explicitly address polyphony and present exclusively monophonic results.

Many variants of the LSTM architecture have been proposed. Perhaps the most well known is the gated recurrent unit (GRU)[20], which constrains the input and forget gates to sum to 1. Mikolov et al. [94] proposed the structurally constrained RNN (SCRN), a simple architecture achieving comparable performance to LSTM. Of most relevance to music, Koutnik et al. [83] proposed the clockwork RNN for explicitly modelling phenomena occurring at multiple timescales by updating different blocks of the hidden state at different periods. Whether these differences matter is not definitive: Greff et al. [64] performed 5400 experiments on eight different architectures and found no significant difference in performance compared to the original LSTM architecture. Nayebi and Vitelli [99] reports LSTM significantly outperform GRUs in music applications.

3.3 Generative modelling of Bach Chorales

One of the first generative models for harmonizing Bach chorales is Bellgard and Tsaing's effective Boltzmann machine model [9]. Their model uses Boltzmann machines to enforce consistency within local contexts. As a result, their model is unable to capture long-range dependencies. Furthermore, they quantize to half-notes and only achieve 1/8 the time resolution of our model.

Allan and Williams [2] used HMMs to harmonize Bach chorales. Their model consists of two separate HMM models: one for generating harmonizations and another for ornamentation.

Their model uses a discrete harmonic encoding of chords for hidden states. In contrast, our model uses an unconstrained continuous hidden state and requires no separate ornamentation step.

Liu and Ramakrishnan [89] applied LSTM to Bach chorales and reports significant gains using RProp instead of BPTT, a technique previously utilized by MELONET[47]. However, they erroneously use a mean squared error training criterion for a classification task, casting doubts on the validity of their experiments.

Brien and Roman [17] compared RNN models for Bach chorales and found clockwork RNNs to yield the lowest validation loss. However, their data format does not permit independent articulation of parts. More importantly, the performance margin between clockwork RNNs and LSTM was very small (6.5 vs 6.75 cross-entropy loss) and their implementation resets the LSTM state when truncating gradients during BPTT, limiting the time-range of learned dynamics to be at most the sequence length.

A MIDI format of the JCB Dataset has become a popular benchmark for RNN research. Pascanu et al. [104] make the observation "that there is no clear winner in the task of polyphonic music prediction" but "in all cases one of the proposed deep RNNs outperformed the conventional, shallow RNN."

Collins et al. [22] is the most recent and relevant work in our area. The authors propose a model called **Racchmaninof** (RAndom Constrained CHain of MArkovian Nodes with INheritance Of Form) and evaluate it on 25 participants with a mean of 8.56 years of formal music training. They impressively find that only 20% of participants performed significantly better than chance.

Supposing, for instance, that the fundamental relations of pitched sound in the signs of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent

Ada Lovelace [16]

4 Automatic composition with deep LSTM

This chapter describes the design and quantitative evaluation of a generative RNN sequence model for polyphonic music. We first construct a training corpus from Bach chorales and investigate the impact of our preprocessing procedure on the corpus. Next, we present a simple frame-based sequence encoding for polyphonic music with many desirable properties.

Using this sequence representation, we reduce the task to one of language modelling and first show that traditional N-gram language models perform poorly on our encoded music data. This prompts an investigation of various RNN architectures, design trade-offs, and training methods in order to build an optimized generative model for Bach chorales.

We conclude this chapter by quantitatively evaluating our final model in test-set loss and training time, and comparing against similar work to establish context.

4.1 Constructing a corpus of encoded Bach chorales scores

We restrict the scope of our investigation to Bach chorales for the following reasons:

1. The Baroque style employed in Bach chorales has specific guidelines and practices [107] (*e.g.* no parallel fifths, voice leading) which can be use to qualitatively evaluate success

2. The large amount of easily recognizable structure: all chorales have exactly four parts consisting of a melody in the Soprano part harmonized by the Alto, Tenor, and Bass parts. Additionally, each chorale consists of a series of **phrases**: "groupings of consecutive notes into a unit that has complete musical sense of its own"' [98] which Bach delimited using fermatas

3. The Bach chorales have become a standardized corpus routinely studied by aspiring music theorists[137]

We use the **Bach-Werke-Verzeichnis** (BWV) [19] indexed collection of the Bach chorales provided by the `music21`[32] Python library as our data source.

4.1.1 Preprocessing

Motivated by transposition invariance in music (see section A.2.4 on page 106) and prior work [96, 44, 51, 49], we first perform **key normalization**. The keys of each score were first analyzed using the Krumhansl Schmuckler key-finding algorithm [84] and then transposed such that the resulting score is C-major for major scores and A-minor for minor scores.

Next, **time quantization** is performed by aligning note start and end times to the nearest multiple of some minimum duration. Our model uses a minimum duration of one semibreve, exceeding the time resolutions of [15, 44] by 2x, [70] by 4x, and [9] by 8x.

We consider only note pitches and durations, neglecting changes in timing (*e.g.* ritardandos), dynamics (*e.g.* crescendos), and stylistic notations (*e.g.* accents, staccatos, legatos). This is comparable to prior work [15, 104] where a MIDI-encoding also lacking this additional notation was used.

An example of the distortion introduced through of our preprocessing steps is provided in fig. 4.1 on the next page in sheet music notation and in piano roll notation on fig. 4.2 on page 48.

Corpus level analysis of preprocessing effects

To assess the effects introduced by key normalization and time quantization, we analyze corpus level statistics related to pitch and duration.

fliang: Trim since we moved a lot to appendix

?? plots a histogram of pitch usage counts before and after key normalization. Notice that the overall range of pitches has increased after key normalization. This can be explained by noting that Bach's chorales were to be performed by vocalists and hence were restricted to use pitches within human voice ranges regardless of key. After transposition, this constraint is no

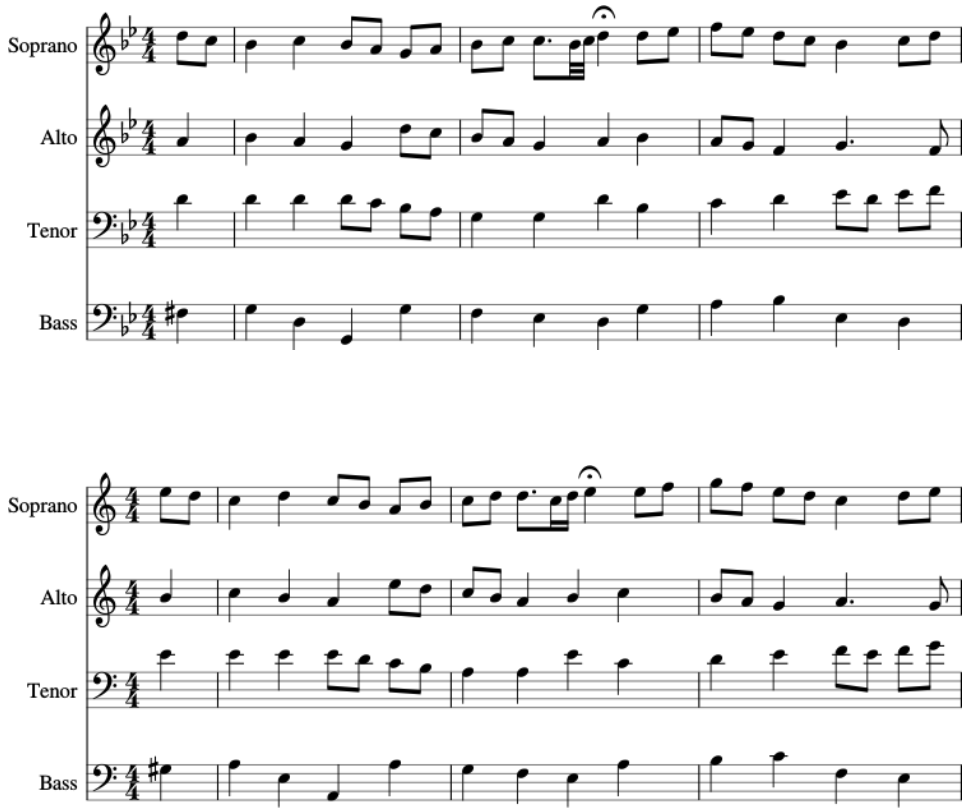


Fig. 4.1 First 4 bars of JCB Chorale BWV 185.6 before (top) and after (bottom) preprocessing. Note the transposition down by a semitone to C-major as well as quantization of the demisemiquavers in the third bar of the Soprano part.

longer be satisfied and we see the appearance of unrealistically low notes (*e.g.* A1) outside the range of even the lowest voice types.

In **??**, we visualize histograms of pitch class usages. As expected, key normalization has increased the usage of pitch classes in the key of C-major / A-minor (*i.e.* those which possess no accidentals) and decreased out of key pitch classes (*e.g.* C#, F#).

fliang: Add tables to quantify, argue that semiquaver quantization is a big win for us

We investigate the effects of time quantization in fig. 4.3, which shows histograms of note duration usages before and after quantization.

fliang: Update plots... are they affected

.

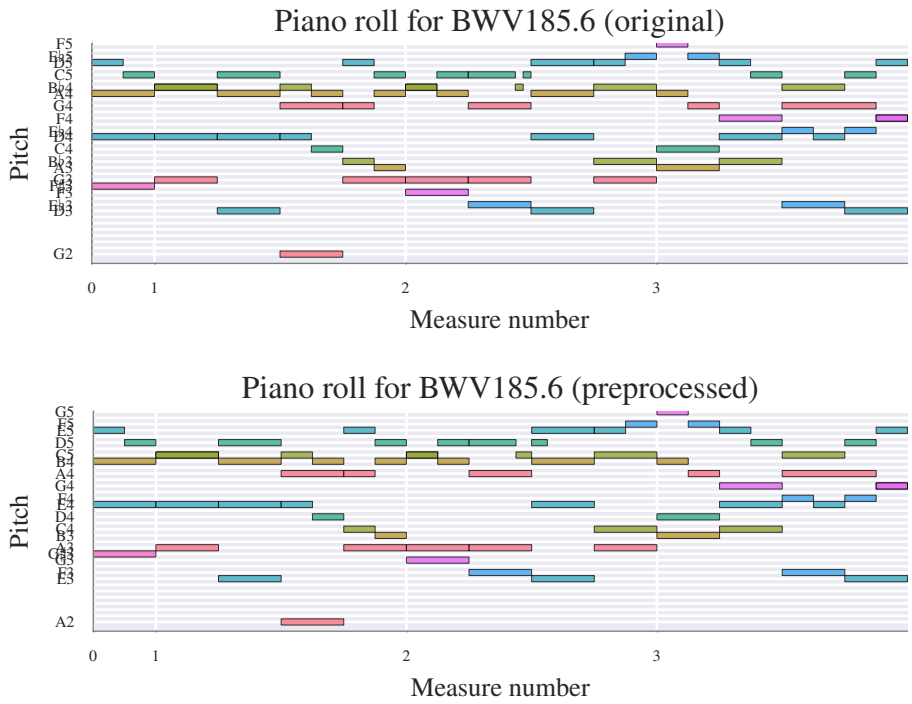


Fig. 4.2 Piano roll representation of the same 4 bars from fig. 4.1 before and after preprocessing. Again, note the transposition to C-major and time-quantization occuring in the Soprano part.

4.1.2 Sequential encoding of musical data

After preprocessing of the scores, our next step is to encode music into a sequence of tokens amenable for processing by RNNs. One design decision is whether the tokens in the sequence are comprised of individual notes (as done in [96, 51, 123]) or larger harmonic units (*e.g.* chords [44, 15], "harmonic context" [2]). This tradeoff is similar to one faced in RNN language modelling where either individual characters or entire words can be used.

In contrast to most language models which operate at the word level, we choose to construct our models at the note level for several reasons. Firstly, the issue of multiple tokens in the sequence corresponding to the same instant of time in the represented music should not be problematic because LSTM have been shown to be able to learn to implement precise timing and counting[57]. Additionally, the use of a note-level encoding partially mitigates the problem of out-of-vocabulary (OOV) tokens in two ways. Besides reducing the potential vocabulary size from $O(128^4)$ possible chords to $O(128)$ potential notes, the model is now able to capture harmonic relationships between notes within the LSTM model weights ($\boldsymbol{W}_{xx}$, $\boldsymbol{W}_{xh}$, $\boldsymbol{W}_{hh}$ in

fliang: reference

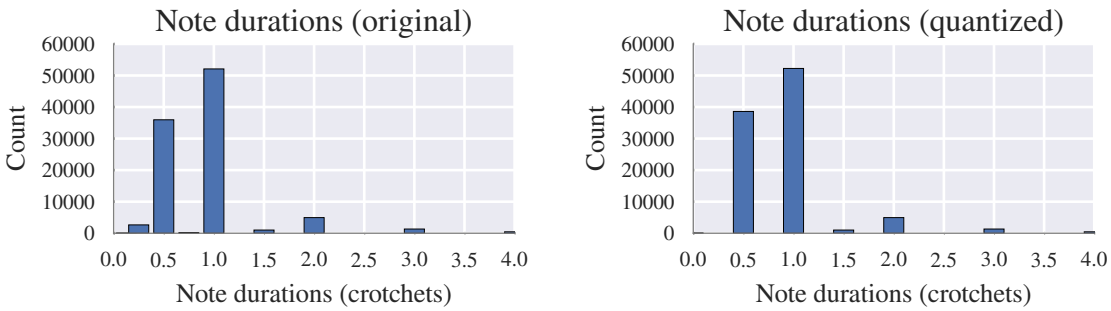


Fig. 4.3 Distribution of note durations over Bach chorales corpus. Quantization has minimal impact because of the high resolution (semiquavers) used.

) and may generalize better to unseen chords. Furthermore, Graves [62] showed comparable performance between LSTM language models that operate on individual characters versus words (perplexities of 1.24 bits vs 1.23 bits per character respectively), suggesting that the choice is not too significant at least for English language modelling.

Similar to [130], we represent polyphonic scores using a localist frame-based representation where time is discretized into constant timestep **frames**. Frame based processing forces the network to learn the relative duration of notes, a counting and timing task which [57] demonstrated LSTM is capable of. Consecutive frames are separated by a unique delimiter ("|||"' in

fliang: Figure of score encoded in text

).

fliang: Add an example encoded score

Each frame consists of a sequence of ⟨Note, Tie⟩ tuples where Note $\in \{0, 1, \cdots, 127\}$ represents the MIDI pitch of a note and Tie $\in \{T, F\}$ distinguishes whether a note is tied with a note at the same pitch from the previous frame or is articulated at the current timestep. A design decision here is the order in which notes within a frame are encoded and consequentially processed by a sequential model. Since chorale music places the melody in the Soprano part, it is reasonable to expect the Soprano notes to be most significant in determining the other parts. Hence, we choose to process the Soprano notes first and order notes in descending pitch within every frame.

The above specification describes our initial encoding. Later in our work

fliang: reference

, we found that this encoding resulted in unrealistically long phrase lengths. Including fermatas (represented by "(.)" in

Table 4.1 Statistics on the preprocessed datasets used throughout our study

Vocabulary size	Total # tokens	Training size	Validation size
108	423463	381117	42346

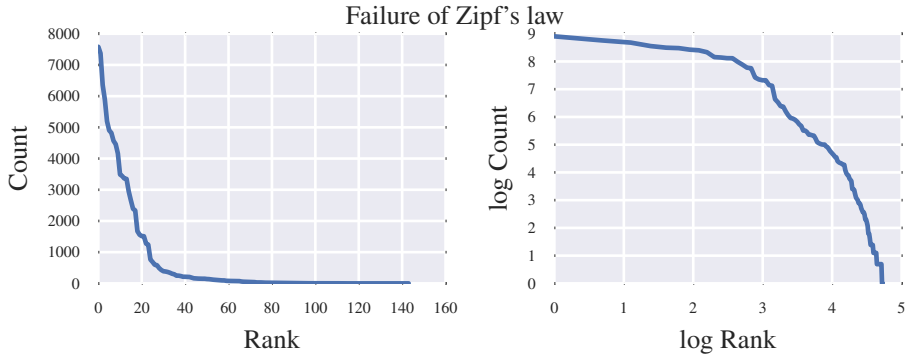


Fig. 4.4 Left: Token frequencies sorted by rank. Right: log-log plot where a power law distribution as predicted by Zipf's law would appear linear.

fliang: Figure of encoded score

, which Bach used to denote ends of phrases, solves this problem.

Finally, for each score a unique start symbol ("START" in

fliang: Figure

) and end symbol ("END" in

fliang: Figure

) are appended to the beginning and end respectively. This causes the model to learn to initialize itself when given the start symbol and allows us to determine when a composition generated by the model has concluded. The vocabulary and corpus size after encoding is detailed in table 4.1. The rank-size distribution of the note-level corpus tokens is shown in fig. 4.4 and confirms the failure of Zipf's law in our data.

fliang: Trim this stuff

Notice that our encoding is sparse: unarticulated notes are not encoded. It is also variable length as anywhere from zero to four (in the case of chorales, more for arbitrary polyphonic scores) notes. Finally, the explicit representation of tied notes vs articulated notes solves the problem where multiple articulations at the same pitch are indistinguishable from a single note with the same duration, an issue present in many prior works [44, 43, 89, 17].

Additionally, notice that our encoding avoids hand-engineered features such as pitch representations which are psychologically-based [96] or harmonically-based [51] [85]. This is

intentional and is motivated by numerous reports [11][12] suggesting that that a key ingredient in deep learning's success is its ability to learn good features from raw data.

4.2 Design and validation of a generative model for music

In this section, we describe the design and validation process leading to our generative model. Unlike many prior models for music data, we intentionally avoid injection of domain-specific knowledge into our model architectures such as distinguishing between chords versus notes [70][96] [44] and explicitly modelling of meter [43] or motifs [47]. Through this fundamental connectionist approach, we aim to minimize biases introduced by prior assumptions and force the model itself to learn musical structure from data.

4.2.1 Training and evaluation criteria

Following [96], we will train the model to predict a distribution distribution over all possible tokens next $\boldsymbol{x}_{t+1}$ given the current token $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$. This is equivalent to setting the target sequence to be the input sequence delayed by one timestep: $\boldsymbol{y}_{1:T-1} = \boldsymbol{x}_{2:T}$ and $\boldsymbol{y}_T = \text{STOP}$.

fliang: Diagram for sequential prediction

.

fliang: Note similarity with language modeling

.

For training criteria, we use the cross-entropy between the predicted distributions $P(\boldsymbol{y}_t|\boldsymbol{h}_t, \boldsymbol{x}_t)$ and the actual target distribution $\delta_{\boldsymbol{y}_t}$.

Note that our training criteria as written in

fliang: reference

uses the actual next token $\boldsymbol{x}_{t+1}$ as the recurrent input, even if the most likely prediction argmax $P(\boldsymbol{x}_{t+1}|\boldsymbol{h}_t, \boldsymbol{x}_t)$ differs. This is is referred to as **teacher forcing**[139] and is motivated by the observation that model predictions may not yet be correct during the early iterations of training. However, at inference the token generated from $P(\boldsymbol{x}_{t+1}|\boldsymbol{h}_t, \boldsymbol{x}_t)$ is reused as the previous input, creating a discrepancy between training and inference. Scheduled sampling [10] is a recently proposed alternative training method for resolving this discrepancy and may help the model better learn to predict using generated symbols rather than relying on ground truth to be always provided as input.

4.2.2 Establishing a baseline with N-gram language models

The encoding of music scores into token sequences permits application of standard sequence modelling techniques from **language modelling**, a research topic within speech recognition concerned with modelling distributions over sequences of tokens (*e.g.* phones, words). This motivates our use of two widely available language modelling software packages, KenLM [68] and SRILM [122], as baselines. KenLM implements an efficient modified Kneser-Ney smoothing language model and while SRILM provides a variety of language models we choose choose to use the Good-Turing discounted language model for benchmarking against.

Both models were developed for applications modelling language data, whose distribution over words which may differ from our encoded music data (see fig. 4.4 on page 50). Furthermore, both are based upon N-gram models which are constrained to only account for short-term dependencies. Hence, we expect RNNs to outperform the results shown in table 4.2 on the facing page.

4.2.3 Description of RNN model hyperparameters

The following experiments investigate deep RNN models parameterized by the following hyperparameters:

1. `num_layers` – the number of memory cell layers
2. `rnn_size` – the number of hidden units per memory cell (*i.e.* hidden state dimension)
3. `wordvec` – dimension of vector embeddings
4. `seq_length`
5. `dropout` – the dropout probability

fliang: Does this need to be diagrammed?

fliang: Justify wordvec embedding

Our model first embeds the inputs $\boldsymbol{x}_t$ into a `wordvec`-dimensional vector-space, compressing the dimensionality down from $|V| \approx 140$ to `wordvec` dimensions. Next, `num_layers` layers of memory cells followed by batch normalization [77] and dropout [120] with dropout probability `dropout` are stacked. The outputs $\boldsymbol{y}_t^{(\texttt{num\_layers})}$ are followed by a fully-connected layer mapping to $|V| = 108$ units, which are passed through a softmax to yield a predictive distribution $P(\boldsymbol{x}_{t+1}|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$. Cross entropy is used as the loss minimized during training.

Table 4.2 Perplexities of baseline N-gram language models on encoded music data

Model Order	KenLM (Modified Kneser-Ney)		SRILM(Good-Turing)	
	Train	Test	Train	Test
1	n/a	n/a	34.84	34.807
2	9.376	8.245	9.420	9.334
3	6.086	5.717	6.183	6.451
4	3.865	4.091	4.089	4.676
5	2.581	3.170	2.966	3.732
6	1.594	2.196	2.002	2.738
7	1.439	2.032	1.933	2.617
8	1.387	2.014	1.965	2.647
9	1.350	2.006	1.989	2.673
10	1.323	2.001	1.569	2.591
11	1.299	1.997	1.594	2.619
12	1.284	2.000	1.633	2.664
13	1.258	1.992	1.653	2.691
14	1.241	1.991	1.682	2.730
15	1.226	1.991	1.714	2.767
16	1.214	1.994	1.749	2.807
17	1.205	1.995	1.794	2.853
18	1.196	1.993	1.845	2.901
19	1.190	1.996	1.892	2.947
20	1.184	1.997	1.940	2.990
21	1.177	1.996	1.982	3.027
22	1.173	1.997	2.031	3.067
23	1.165	1.997	2.069	3.101
24	1.159	1.998	2.111	3.135
25	1.155	2.000	2.156	3.170

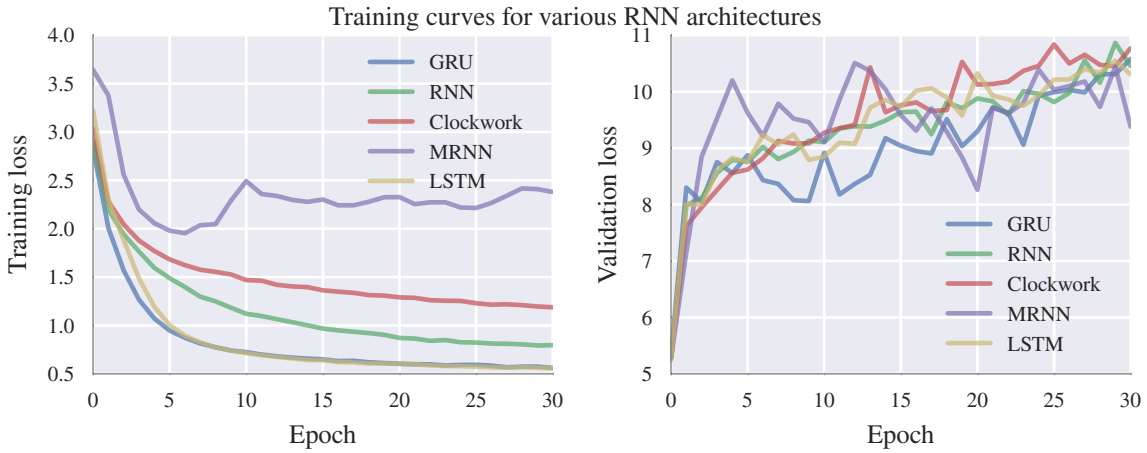


Fig. 4.5 LSTM and GRUs yield the lowest training loss. Validation loss traces show all architectures exhibit signs of significant overfitting

Models were trained using the Adam normalization [82] and an initial learning rate of 2×10^{-3} decayed by 0.5 every 5 epochs. The back-propogation through time gradients were clipped at t[103] and truncated after `seq_length` time steps. We use a mini-batch size of 50.

4.2.4 Comparison of memory cells on music data

To rapidly compare a large number of memory cell implementations, we leveraged `theanets`[1]: a Python software library for high-level specification of neural network models. Figure 4.5 shows the results of exploring a range of RNN memory cell implementation and holding `num_layers=1`, `rnn_size=130`, `wordvec=64`, and `seq_length=50` constant. Unlike later models, none of these models utilized dropout or batch normalization. We used a batch size of 50 and configured the clockwork RNN [20] with 5 equal-sized hidden state blocks with update periods $(1, 2, 4, 8, 16)$.

Figure 4.5 shows that while all models achieved similar validation losses, LSTM and GRUs trained much faster and achieved lower training loss. Since Zaremba [143] find similar empirical performance between LSTM and GRUs and Nayebi and Vitelli [99] observe LSTM outperforming GRUs in music applications, we choose to use LSTM as the memory cell for all following experiments.

The increasing validation loss over time in fig. 4.5 is a red flag suggesting that overfitting is occuring. This observation motivates the exporation of dropout regularization in section 4.2.5.

[1] https://github.com/lmjohns3/theanets

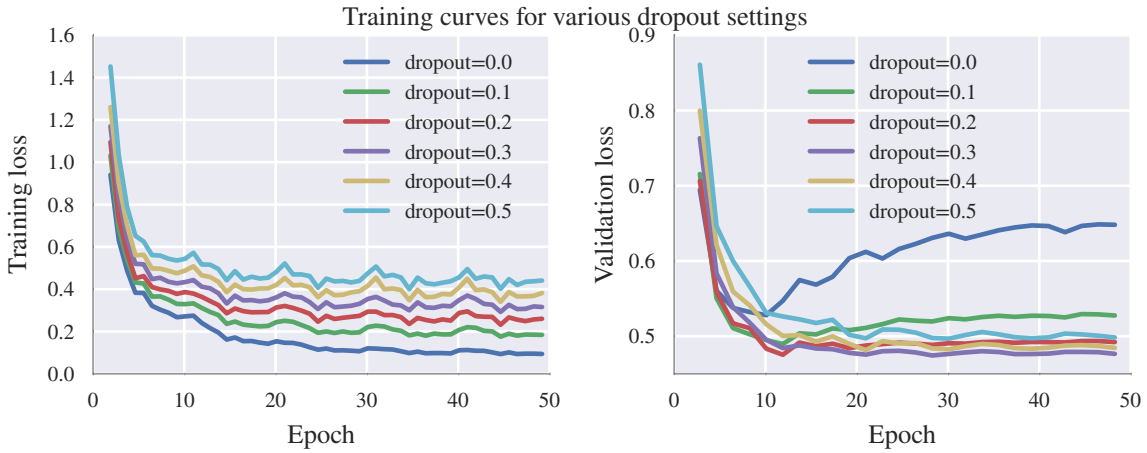


Fig. 4.6 Dropout acts as a regularizer, resulting in larger training loss but better generalization as evidenced by lower validation loss. A setting of `dropout=0.3` achieves best results for our model.

4.2.5 Optimizing the LSTM architecture

After settling on LSTM as the memory cell, we conducted remaining experiments using the `torch-rnn` Lua software library. Our switch was motivated by support for GPU training, dropout, and batch normalization.

Dropout regularization improves validation loss

fliang: Is it still worth it to do some batch normalization experiments?

The increasing validation errors in fig. 4.5 on page 54 prompted investigation of regularization techniques. In addition to adding batch normalization, a technique known to reduce overfitting and accelerate training [77], we also investigated the effects of different levels of dropout by varying the `dropout` parameter.

The experimental results are shown in fig. 4.6. As expected, dropout acts as a regularizer and reduces validation loss from 0.65 down to 0.477 (when `dropout=0.3`). Training loss has slightly increased, which is also unexpected as application of dropout during training introduces additional noise into the model.

Overall best model

We perform a grid search through the following parameter grid:

- `num_layers` $\in \{1, 2, 3, 4\}$

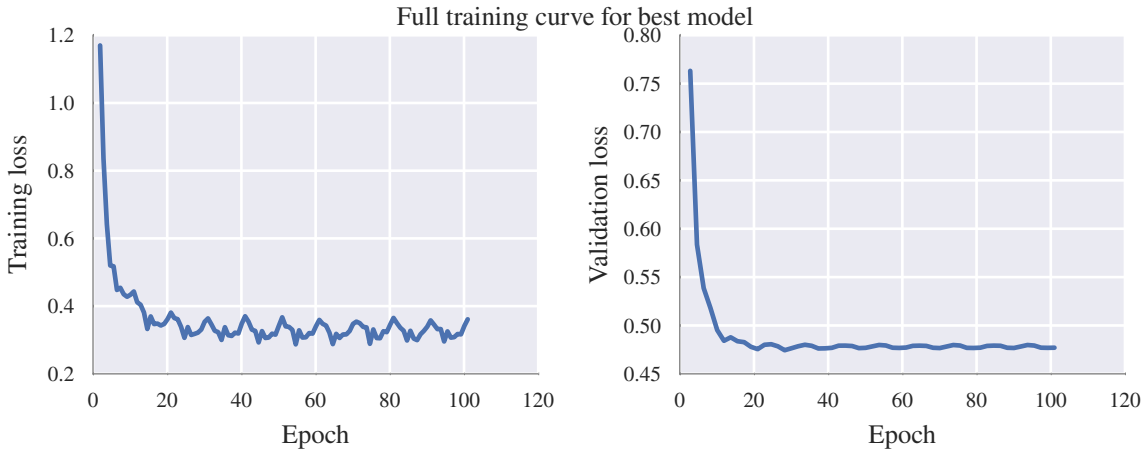


Fig. 4.7 Training curves for the overall best model. The periodic spikes correspond to resetting of the LSTM state at the end of a training epoch.

- `rnn_size` $\in \{128, 256, 384, 512\}$
- `wordvec` $\in \{16, 32, 64\}$
- `seq_length` $\in \{64, 128, 256\}$
- `dropout` $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$

A full listing of results is provided in **??** on page ??.

The optimal hyperparameter settings within our grid was found to be `num_layers` $= 3$, `rnn_size` $=$, `wordvec` $= 32$, `seq_length` $= 128$ `dropout` $= 0.3$. Such a model achieves 0.324 and 0.477 cross entropy losses on training and validation corpuses respectively. Figure 4.7 plots the training curve of this model and shows that training converges after only 30 iterations.

To confirm local optimality, we perform perturbations about our final hyperparameter settings in **??**–**??**.

Consistent with prior work [126, 80], the table 4.3 shows that GPU training resulted in training speedups of more than 800% over CPU training.

Sensitivity to network structure: `num_layers` and `rnn_size`.

- Larger `rnn_size` leads to higher capacity and lower training loss
  - Presents as overfitting on validation, where the lowest capacity model `rnn_size` appears to be improving in generalization while others are flat/increasing
- Training curves about the same wrt `num_layers`, validation curves have interesting story

Table 4.3 Timing results comparing CPU and GPU training of the overall best model (section 4.2.5 on page 55)

	Single Batch		30 Epochs (seconds)
	mean (sec)	std (sec)	(minutes)
CPU	4.287	0.311	256.8
GPU	0.513	0.001	28.5

  - Depth matters: small 64 and 128 hidden unit RNNs saw improvements up to 0.09
  - Unsurprising, Pascanu et al. [104] already observed deep RNNs dominated polyphonic modelling.
  - Expressivity gained from depth furthers overfitting: 256 hidden unit RNN has some of the best validation performance at depth 1 but is the worst generalizing model for depths 2 and 3 even though training loss is low

- `rnn_size=128` found to be best generalizing, optimized at `num_layers=2`: will continue with these settings

Sensitivity to network inputs: `seq_length` and `wordvec`

- Training losses are about the same across all `wordvecs`
- Validation losses suggest that increasing `seq_length` important for good performance

  fliang: investigate further

- `wordvec=128` overfits for all cases, the other two depend on `seq_length` and vary an order of magnitude smaller than the performance gains from increasing `seq_length`

4.3 Results

As done by [8, 15], we quantitatively evaluate our models using cross entropies and perplexities on a 10% held-out validation set.

Allan and Williams [2] achieve cross-entropy losses of $2.79-2.80$ on unseen test-set scores respectively for their "harmonic skeleton" sub-task. This task involves predicting a sequence of one of 81 harmonic symbols, which may be interpreted as equivalence classes of chords. Even after applying Viterbi decoding to find globally optimal sequences, their HMM models achieve cross entropies of $0.84 - 0.87$ on training-set scores (Table 5.2 in Allan and Williams [2]).

Our best model achieves cross-entropy losses of 0.323 on training data and 0.477 on held-out test data (**??** on page ??), corresponding to a training perplexity of 1.251 bits and a test perplexity of 1.391. This is more than 0.6 bits lower than any test perplexity obtained by the N-gram models compared in table 4.2 on page 53. This result is expected and shows that the greater modelling capacity provided by RNNs is useful for encoded music score data.

fliang: Compare [17]

.

The best models investigated in Boulanger-Lewandowski, Vincent, and Bengio [15] achieved -5.56 log likelihood and 33.12% accuracy on the symbolic prediction task. Note that the log likelihood is significantly lower than the -0.323 log likelihood implied by our model's cross-entropy loss. This is because the two results are not comparable as they utilize different input encodings with varying vocabulary sizes.

fliang: Compare on pitch/pitch class usage, note durations, meter, lengths of compositions

4.4 Other applications

Scoring things as "Bach-like", model for expectation by using the probability.

We find ourselves in front of an attempt, as objective as possible, of creating an automated art, without any human interference except at the start, only in order to give the initial impulse and a few premises, like in the case of…nothingness in the Big Bang Theory

Hoffmann [73]

5 Analysis of musical concepts learned by the model

5.1 Investigation of neuron activation responses to applied stimulus

One method for gaining insight into what a connectionist model has learned is to apply some stimulus and measure neuron activations at different layers.

We use as stimulus the music score shown in fig. 5.1, which has been preprocessed according to

fliang: cite preprocessing

. To aid in relating neuron activities back to music theory, chords are annotated with Roman numerals obtained using `music21`'s automated analysis.

fliang: This stuff is now in the appendx

In **??** we visualize the network activations as the stimulus is sequentially applied. Note that as a consequence of the variable-length encoding format described in

fliang: ref

, the horizontal axis (number of tokens processed) does correspond directly to time. Rather, time is advanced one frame every time a chord boundary delimiter symbol is output.

5.1.1 Pooling over frames

In order to align and compare the activation profiles with the original score, all the activations occuring in between two chord boundary delimiters must be combined. This aggregation of neuron activations from higher resolution (*i.e.* note-by-note) to lower resolution (*i.e.* frame-by-frame) is reminiscent of pooling operations in convolutional neural networks

fliang: cite

. Motivated by this observation, we introduce the method for pooling an arbitrary number of token-level activations into a single frame-level activation.

Let $\boldsymbol{z}^{(l)}_{t_m:t_n}$ denote the activations of layer l from the t_mth input token $\boldsymbol{x}_{t_m}$ to the t_nth input token $\boldsymbol{x}_{t_n}$. Suppose that $\boldsymbol{x}_{t_m}$ and $\boldsymbol{x}_{t_n}$ are respectively the mth and nth chord boundary delimiters within the input sequence. Define the **max-pooled frame-level activations** $\tilde{\boldsymbol{z}}^{(l)}_n$ to be the element-wise maximum of $\boldsymbol{z}^{(l)}_{t_m:t_n}$, that is:

$$\tilde{\boldsymbol{z}}^{(l)}_n := \left[\max_{t_m<t<t_n} \boldsymbol{z}^{(l)}_{t,1}, \quad \max_{t_m<t<t_n} \boldsymbol{z}^{(l)}_{t,2}, \quad \cdots, \quad \max_{t_m<t<t_n} \boldsymbol{z}^{(l)}_{t,N^{(l)}} \right]^\intercal \tag{5.1}$$

where $\boldsymbol{z}^{(l)}_{t,i}$ is the activation of neuron i in layer l at time t and $N^{(l)}$ is the number of neurons in layer l. Notice that the pooled sequence $\tilde{\boldsymbol{z}}$ is now indexed by frames rather than by tokens and hence corresponds to time-steps.

We choose to perform max pooling because it preserves the maximum activations of each neuron over the frame. While pooling methods (*e.g.* sum pooling, average pooling) are possible, we did not find significant differences in the visualizations produced.

The max-pooled frame-level activations are shown in fig. 5.2. As a result of pooling, the horizontal axis can be aligned and compared against the stimulus fig. 5.1. Notice the appearance of vertical bands corresponding to when a chord/rest is held for multiple frames. In particular, the vector embedding corresponding to rests (*e.g.* near frames 30 and 90 in fig. 5.2 top) are sparse, showing up as white smears not only in the embedding layer but on all LSTM memory cells.

5.1.2 Probabilistic piano roll: likely variations of the stimulus

fliang: Revise: the piano roll reference is removed

The bottom panel in fig. 5.2 shows the model's predictions for tokens in the next frames, where the tokens are arranged according to (arbitrary) index within the vocabulary. As the tokens correspond to pitches, they can be sorted according to pitch to reconstruct a **probabilistic piano roll**[43] consisting of the model's sequence of next-frame predictions as it processes the input.

fliang: Revise: the piano roll reference is removed

Notice that the probabilistic piano roll in fig. 5.3 closely resembles the stimulus. This is unsurprising because the recurrent inputs are taken from the stimulus rather than sampled from the model's predictions (a.k.a. [139]), so a model which predicts to only continue holding its input would produce a probabilistic piano roll identical to the stimulus delayed by one frame.

Two interesting rows of fig. 5.3 are the rows corresponding to frame delimiters (fourth from top, "|||") and fermatas (third from top "(.)"). Notice that the predictions for chord delimiters are particularly strong during rests. This is because rests are encoded as empty frames, so the large probability values indicate that the model has learned to prolong periods of rests. At the end of rest periods, the model tends to assign probability across a wide range of notes, consistent with the intuition that the possible notes occuring directly after a rest is less constrained than

fliang: cite the intuition?

those occuring in the middle of a phrase. Finally, notice that the probability assigned to fermatas is larger near the ends of phrases, suggesting that the model has successfully learned the concept of phrasing within music.

The probabilistic piano roll can be interpreted as variations on the stimulus which the model finds likely and may serve as a useful computational tool for generating likely chorale variations.

5.1.3 Neurons specific to musical concepts

Research in convolutional networks has shown that individual neurons within the network oftentimes specialize and specifically detect certain high-level visual features

fliang: Cite deconvolution

. Extending the analogy to musical data, we might expect certain neurons within our learned model to act as specific detectors to certain musical concepts.

To investigate this further, we look at the activations over time of individual neurons within the LSTM memory cells. Our results confirm our hypothesis: we discover certain neurons whose activities are correlated to specific motifs, chord progression, and phrase structures. The activity profiles of these neurons are shown in fig. 5.4.

For notational clarity, we will use the ordered tuple (l, i) to refer to the ith neuron in layer l.

The first three neurons ((1, 64), (1, 138), (1, 207)) shown in the 2nd to 4th panel from top of fig. 5.4 effectively behave like cadence detectors. While they all exhibit activity when the stimulus contains V chords (*i.e.* G-major). (1, 64) and (1, 138) are both specific to perfect cadences (*i.e.* $V--I$ chord progressions) used to conclude phrases and differ only in the chord inversions which they are most sensitive to. In contrast, (1, 207) only exhibits activity for the V chord associated with the imperfect cadences near frames 150 and 180.

The next two neurons in fig. 5.4, (1, 87) and (1, 151), act as motif detectors. Activity in (1, 151) peaks when a $vi - vii - vi$ progression is present in the stimulus. (1, 87) exhibits large spikes on $I--V--I$,

(2, 37) exhibits less specificity, but has large spikes right before the IV chord in $I--IV$ chord progressions.

(2, 82) peaks at the top of an ascending harmonic progressions, right before a descending major scale is to follow.

(2, 243) is specific to $v--vi$ progressions, with large spikes occuring at the $v--vi$ progressions near frames 55, 120, 130, and a lower intensity spike at 170. Some activity is also observed for the $V--vi$ around frame 230 despite the first chord being a major mode V rather than minor v.

fliang: Add mark's analysis and credit him

fliang: This refutes criticisms of black boxness of approach, play it up

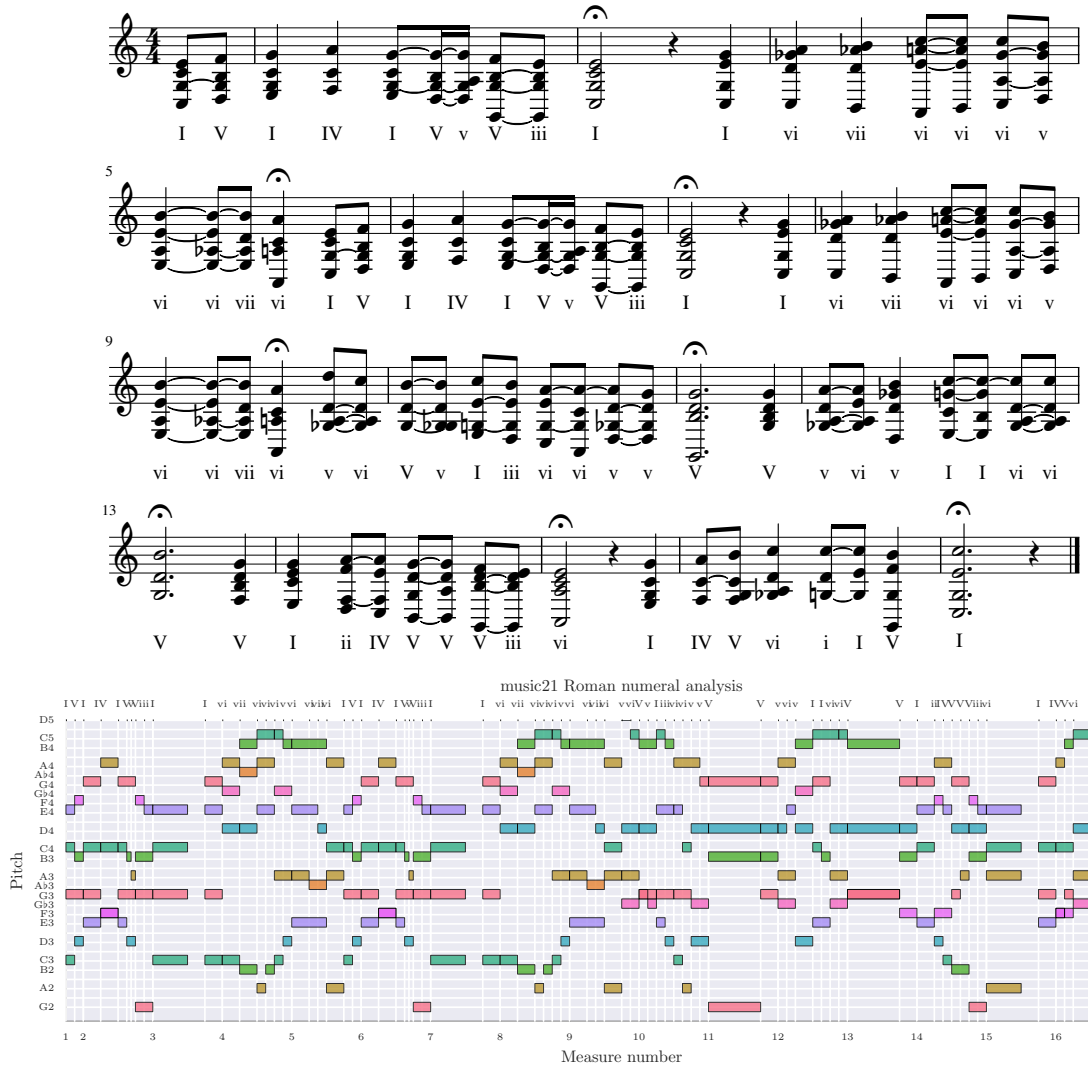


Fig. 5.1 *Top*: The preprocessed score (BWV 133.6) used as input stimulus with Roman numeral analysis annotations obtained from `music21`; *Bottom*: The same stimulus represented on a piano roll

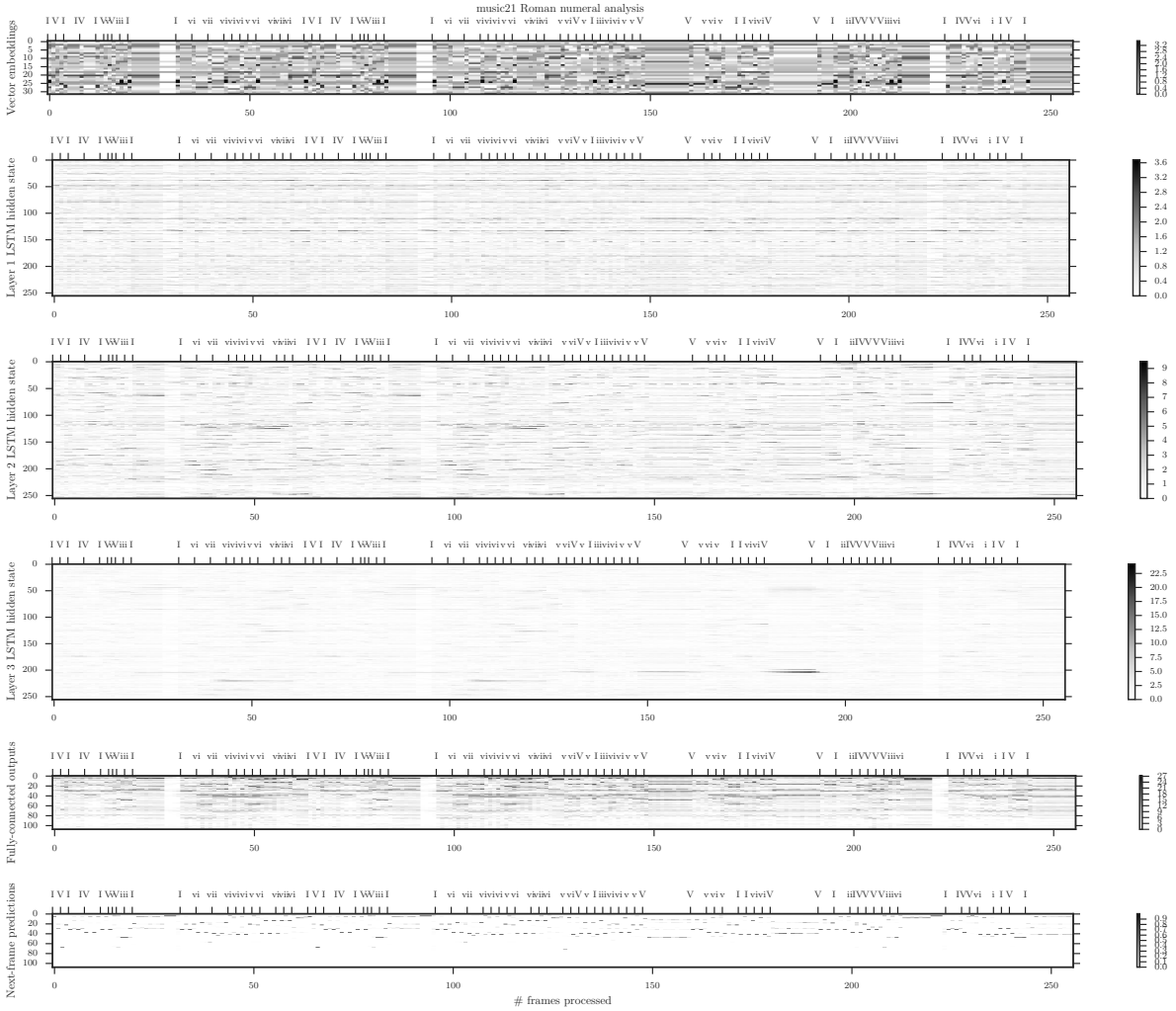


Fig. 5.2 Neuron activations over time pooled over frames

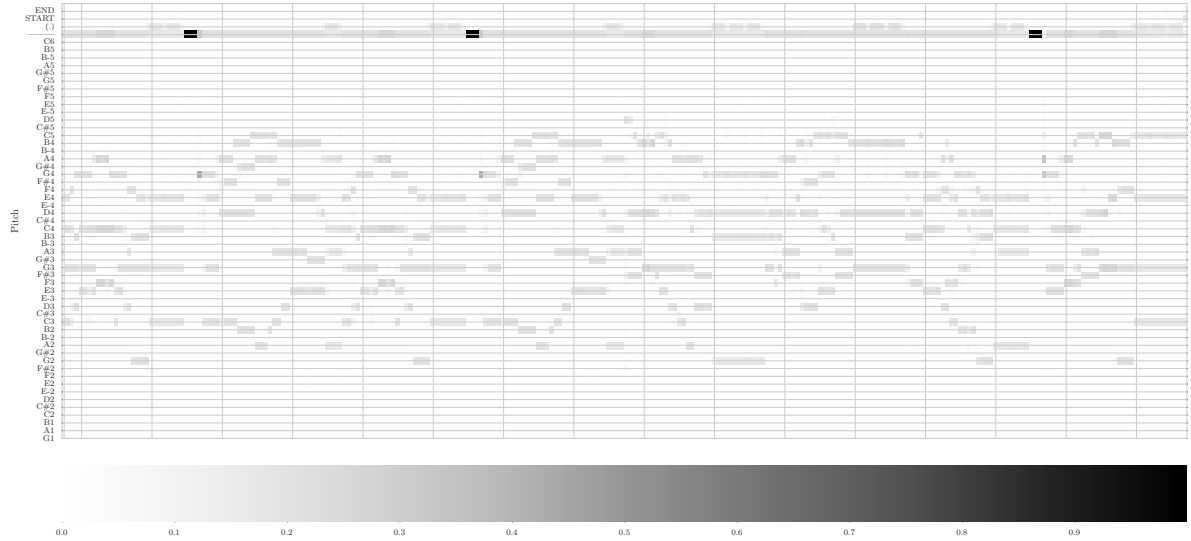


Fig. 5.3 Probabilistic piano roll of next note predictions. Note the strong predictions for fermatas near ends of phrases and the uncertain predictions immediately after long periods of rest.

Fig. 5.4 Activation profiles of neurons within our model which have learned high-level musical concepts

6

Chorale harmonization

fliang: Talk about how correct harmonization is equivalent to conditioning on future hidden state over all possible $\boldsymbol{h}$ trajectories passing through it. Intractable conditioning, so we approximate by neglecting to constrain (*i.e.* don't account for future at all) and instead do teacher forcing. Hope that teacher forcing induces the hidden state to go in a reasonable trajectory, but results show otherwise

Unlike automatic composition, in harmonization tasks we are given the entire sequence of notes for one or more parts. As one of the parts is now fixed, the model is no longer able to freely compose and harmonic deviations from the fixed parts will result in dissonances and conflicting expectations. This lack of accountability for future expectations is one of the failure modes of our models. One potential method for mitigating this is bidirectional LSTM [63], which account for both the future and prior contexts. However, a bidirectional LSTM cannot be sequentially sampled to perform automatic composition.

6.1 Background

A chorale consists of four parts: soprano, alto, tenor, and bass. Chorale harmonization involves producing the alto, tenor, and bass parts given a fixed soprano melody. As described by Walter Piston [107]:

> True harmonisation, then, means a consideration of the alternatives in available chords, the reasoned selection of one of these alternatives, and the tasteful arrangement of the texture of the added parts with due regard for consistency of style

The Baroque style employed by Bach has specific guidelines such as disallowing parallel fifths and parallel octaves as well as considerations for voice leading [107].

For a music student studying chorale harmonization, a common pedagogical exercise [36][107] is a sequence of tasks increasing in difficulty:

1. Providing either alto or tenor given fixed soprano and bass
2. Providing both alto and tenor parts given fixed soprano and bass
3. Harmonization: provide all remaining parts given only the soprano line
4. Composition: provide all four parts with nothing given in advance

There are no definitive formalization of the harmonization process, making evaluation difficult. Attenpts to formalize the process using Shenkerian structural analysis [115] and symbolic methods such as generative grammars [86][141] exist, but involve human analytical process.

6.2 Harmonizing

For chorale harmonization, we are interested in predicting the notes for a part given the other parts. Concretely, suppose we wish to predict a $L \in \mathbb{N}$ length sequence $w_{1:L}$. Let $\alpha \subset [1, T]$ be a multi-index, $\alpha^c \coloneqq [1, T] \setminus \{\alpha\}$, and w_α the tokens corresponding to the given parts. We are interested in finding

$$w^*_{1:L} = \underset{w_{1:L}}{\operatorname{argmax}}\, P(w_{1:L} | w_\alpha) \tag{6.1}$$

We can constrain the set of candidate sequences by first noting any solution $\hat{w_{1:L}}$ must satisfy $\hat{w}_\alpha = w_\alpha$. In Hinton and Sejnowski [71], this is referred to as "clamping" the generative model. Next, we can "fill in" the unconstrained indices in the sequence by greedily sample from our generative model to arrive at an approximate solution:

$$\hat{w}_t = \begin{cases} w_{\alpha_t} & \text{if } t \in \alpha \\ \operatorname{argmax}_{w_t} \hat{P}(w_t | \hat{w}_{1:t-1}) & \text{otherwise} \end{cases} \tag{6.2}$$

where the hat on the previous words $\hat{w}_{1:t-1}$ indicates that they are set equal to the actual previous argmax choices.

This solution is approximate because while the factorization

$$P(w_{1:L}) = \prod_{t=1}^{L} P(w_t|w_{1:t-1}) \tag{6.3}$$

is true and justifies our model, the factorization

$$P(w_{1:L}|w_\alpha) = \prod_{t=1}^{L} \hat{P}(w_t|w_{1:\hat{t-1}}) \tag{6.4}$$

does not hold. Some primary criticisms include

- Modeling capacity limits for RNNs: the model $\hat{P}$ may not be able to fully express the true distribution P (*e.g.* if P is non-Markovian)
- Greedy sequential selection: it is possible that the greedy argmax at each time without accounting for future constraints on sequences $(w_{\alpha_{t'}})_{t'>t}$ leads to a solution with sub-optimal joint probability
- Assumption that prior selections $w_{1:\hat{t-1}}$ optimize $P(w_t|w_{1:t-1})$: the model $\hat{P}$ is trained on data which assumes all prior inputs have beeen ground truth. It has been shown

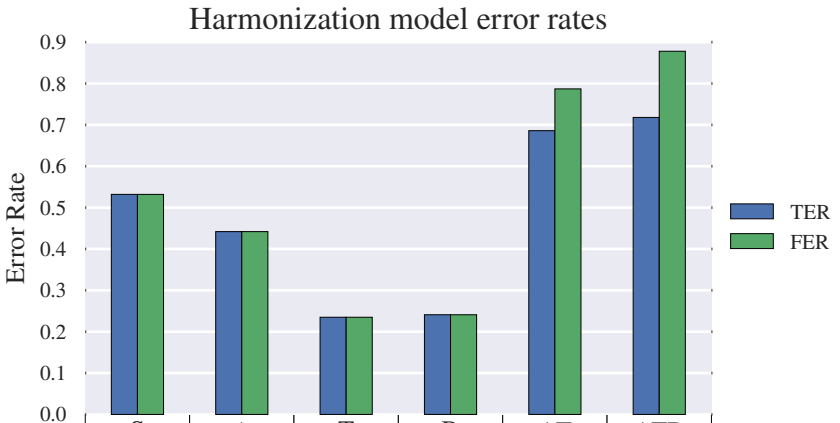
  fliang: mikolov

  that such an assumption can lead to very sensitive hidden state dynamics which are not robust to errors (*i.e.* when $w_{1:\hat{t-1}}$ contain errors).

Beam search is one way to mitigate the effects of greedy selection: our current method is equivalent to a beam search with width one. Maintaining N-best hypotheses using a lattice-based framework such as in Liu et al. [90] would allow the model to partially recover from mistakes made during greedy action selection.

Furthermore, another potential issue with the proposed model is a discrepancy between the inputs provided during training (which are taken from actual data sequences regargless of model predictions) and sampling (where the inputs are generated by the model at the previous timestep). To resolve this discrepancy, Bengio et al. [10] proposed **scheduled sampling** as a curriculum learning strategy gradually transitioning sequence models to rely on their prior predictions as inputs and thereby learn state dynamics more resilient to erroneous inputs.

Despite these limitations, implementation of greedy action selection is still valuable because it forms the basis for more sophisticated lattice-based search methods as well as provides a baseline for comparing performance against.



	S	A	T	B	AT	ATB
TER	0.532	0.442	0.235	0.241	0.686	0.718
FER	0.532	0.442	0.235	0.241	0.787	0.878

Fig. 6.1 Error rates for harmonization tasks

6.3 Datasets

We create datasets where one or more parts are masked:

- A single voice: Soprano (S), Alto (A), Tenor (T), or Bass (B)
- The middle two voices (AT)
- All voices except Soprano (ATB), aka **harmonization**

Of particular interest is the AT dataset. Bach oftentimes only wrote the Soprano and Bass parts of a piece, leaving the middle parts to be filled in by students. Our networks performance on this task can be used as a benchmark for an easier task. Another intersting configuration is ATB, which corresponds to harmonizing a given melody (*i.e.* Soprano line) and can be compared against prior work such as Allan and Williams [2].

6.4 Results

We are unable to compare against Allan and Williams [2] because their model consists of multiple domain specific subtasks (*e.g.* harmony and chord skeletons, ornamentation) and because they only consider the ATB case.

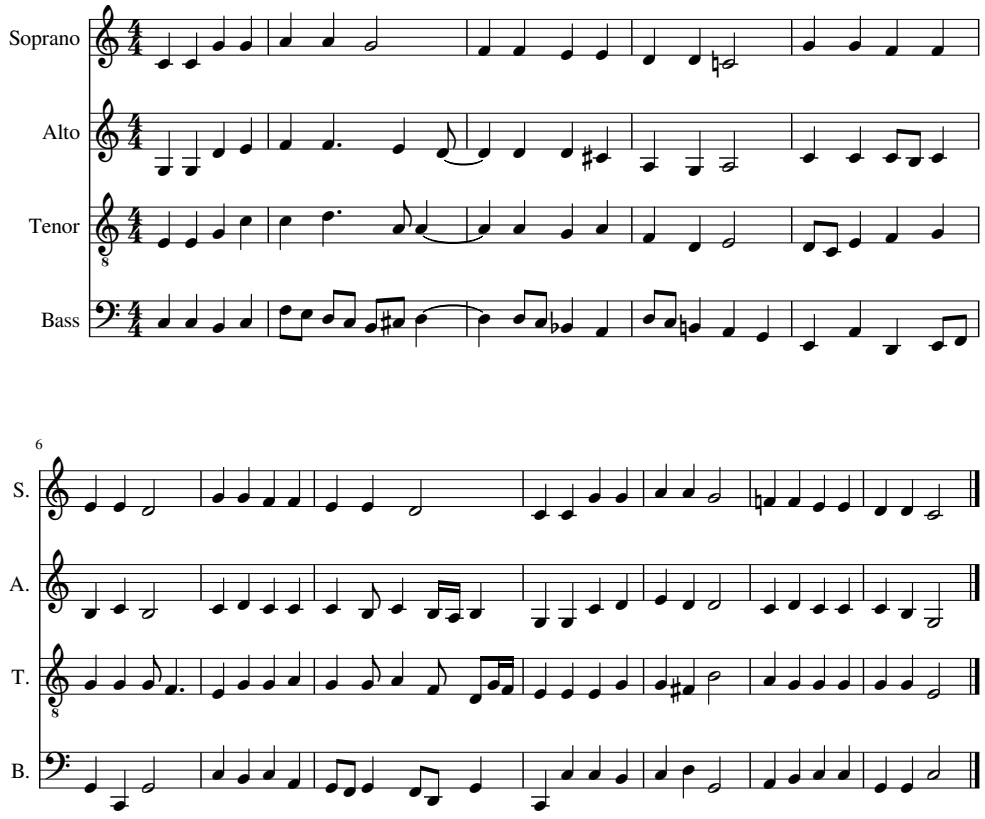


Fig. 6.2 Twinkle-twinkle soprano melody, ATB harmonized by BachBot

6.4.1 Harmonizing popular tunes with BachBot

In addition to being able to harmonize Bach chorales, we found that BachBot was capable of generating Baroque acompanyments to a variety of pieces. fig. 6.2 shows BachBot's proposed ATB harmonization for *Twinkle Twinkle Little Star*.

fliang: EXPERIMENT: Given the head, fill in the rest.

7

Large-scale subjective human evaluation

Many prior studies [42, 2, 15, 91] evalute their success using either log likelihood or self-assessing some generated samples.

[106] addresses difficulty in quantitative evaluation, suggesting the use of a learned critic in a manner similar to generative adversarial networks [61]. In a later report, [105] attribute difficulty in evaluation due to lack aim: algorithmic composition, design of compositional tools, and computational modelling of musical styles or music cognition all have different motivations and should thus be evaluated differently.

Following advice of [105], we identify our research motivation as building "computational models" for musical style and cognition. Specifically, our goal is to build a model which composes music in a style that is perceptually indistinguishable from Bach. To evaluate our results, we adapt Alan Turing's proposed "Imitation Game" [133] and carry out a large-scale musical Turing test.

[5] criticizes a musical Turing test as providing little data about how to improve the system, suggesting that listener studies using music experts may be more insightful.

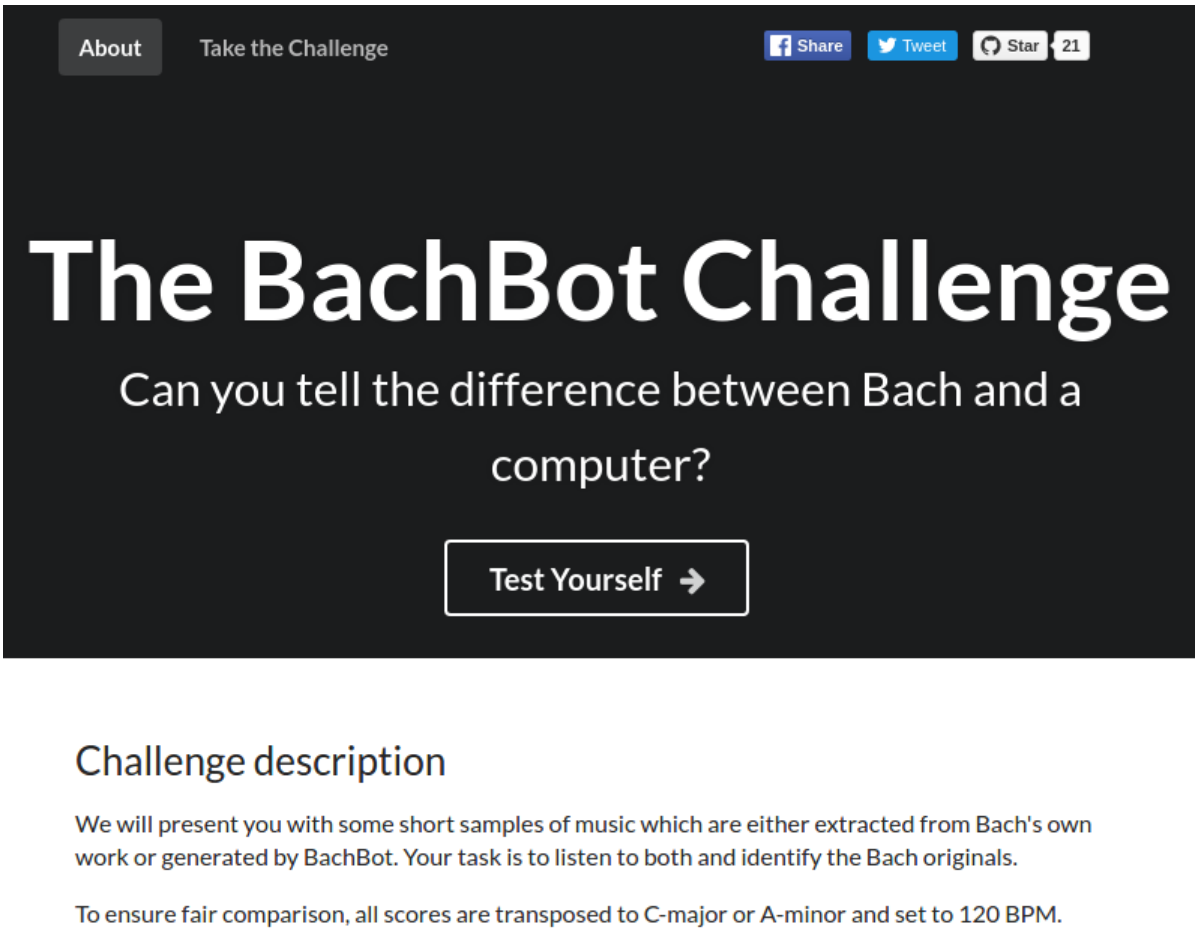


Fig. 7.1 The first page seen by a visitor of http://bachbot.com

7.1 Evaluation framework design

7.1.1 Software architecture

The frontend utilizes React and Redux, allowing us to collect fine-grained user action data. Azure App Service is used to host an Express web-service which randomizes experimental questions and collects responses. The data is stored to Azure Data Storage and processed in batch MapReduce using Azure HDInsight.

7.1.2 User interface

The landing page for http://bachbot.com/ is shown in fig. 7.1.

Clicking "Test Yourself" redirects the participant to a user information form (fig. 7.2) where users self-report their age group prior music experience into the categories shown.

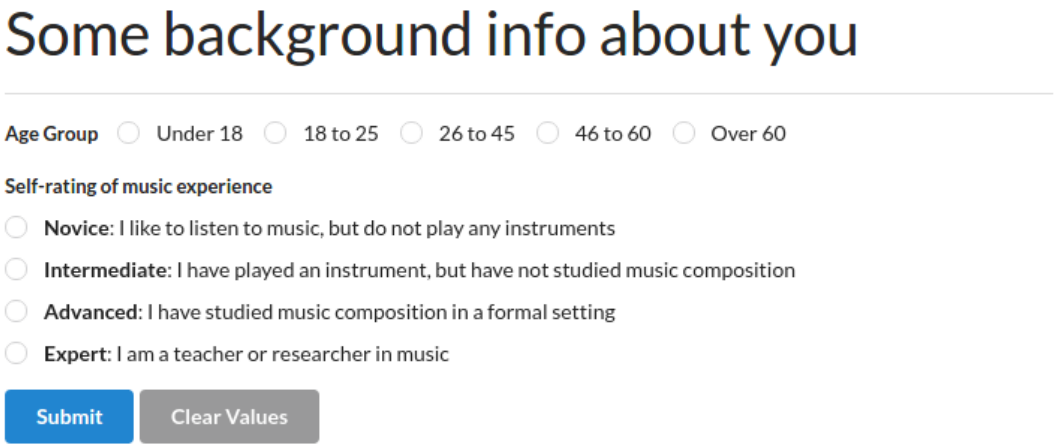


Fig. 7.2 User information form presented after clicking "Test Yourself"

After submitting the background form, users were redirected to the question response page shown in fig. 7.3. This page contains two audio samples, one extracted from Bach and one generated by BachBot, and users were asked to select the sample which sounds most similar to Bach. Users were asked to provide five consecutive answers and then the overall percentage correct was reported.

7.1.3 Question generation

Questions were generated for both harmonizations (using the same abbreviations as defined in

fliang: ref

) as well as original compositions (denoted SATB as all parts are generated). For each question, a random chorale was selected without replacement from the corpus and paired with a corresponding harmonization. SATB sampls were paired with chorales randomly sampled from the corpus. The five question answered by any given participant were comprised of one S/A/T/B question chosen at random, one AT question, one ATB question, and two original compositions. See table 7.1 for details.

7.1.4 Promoting the study

We promoted the study through solely through social media and personal contacts. Participation was voluntary and growth was organic; we did not solicit any paid responses or advertis-

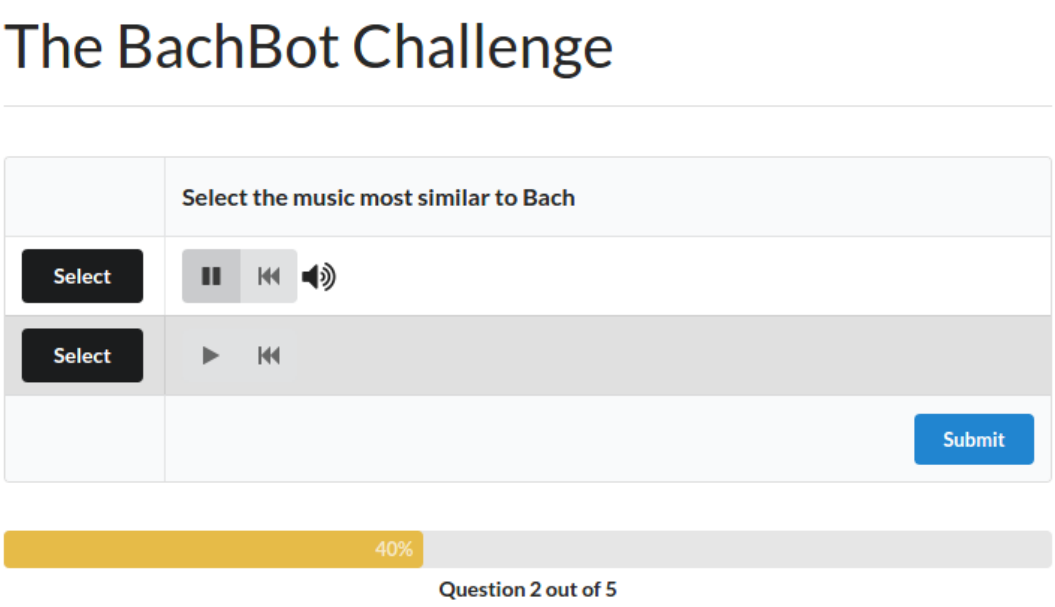


Fig. 7.3 Question response interface used for all questions

DATE ing. We found that 52.1%of participants were referred from social media, 4.8% through other websites and blogs, 2% through search engine results, and the remaining 41.2% had directly accessed bachbot.com,

7.2 Results

7.2.1 Participant backgrounds and demographics

fliang: Update with most recent results before submitting

Question type	# questions available	Expected # responses per participant
S	2	0.25
A	2	0.25
T	2	0.25
B	2	0.25
AT	8	1
ATB	8	1
SATB	12	2

Table 7.1 Composition of questions on http://bachbot.com

We recieved a total of 695participants from different countries. After selecting only the first response per IP address and filtering down to participants whom had listened to both choices in every question, we are left with 657participants answering 5 questions each to yield a total of 3285responses.

As evidenced by fig. 7.4 on the next page, our participant is diverse and includes participants from six different continents. fig. 7.5 on page 79 shows that while the majority of our participants are between 18 – 45and have played an instrument, more than 23.7%have either formally studied or taught music theory.

UPDATE

UPDATE

UPDATE

UPDATE

UPDATE

UPDATE

7.2.2 BachBot's performance results

fliang: Average time per response, average number plays/replays

fliang: fig. 7.6 suggests performance is weakest on harmonizations. Unsurprising because we only do 1-best and don't account for future. Bidirectional LSTM or N-best lattice search (reference marcin) would do better

fig. 7.6 shows the performance of BachBot on various question types. It shows that 59%

fliang: VERIFY LAST

of participants could correctly identify original Bach from BachBot's generated music. As the baseline method of randomly guessing between the two choices in fig. 7.3 achieves 50%, our findings suggest that **the average participant has only a** 9%

fliang: VERIFY LAST

better chance than randomly guessing when distinguishing Bach from BachBot correctly.

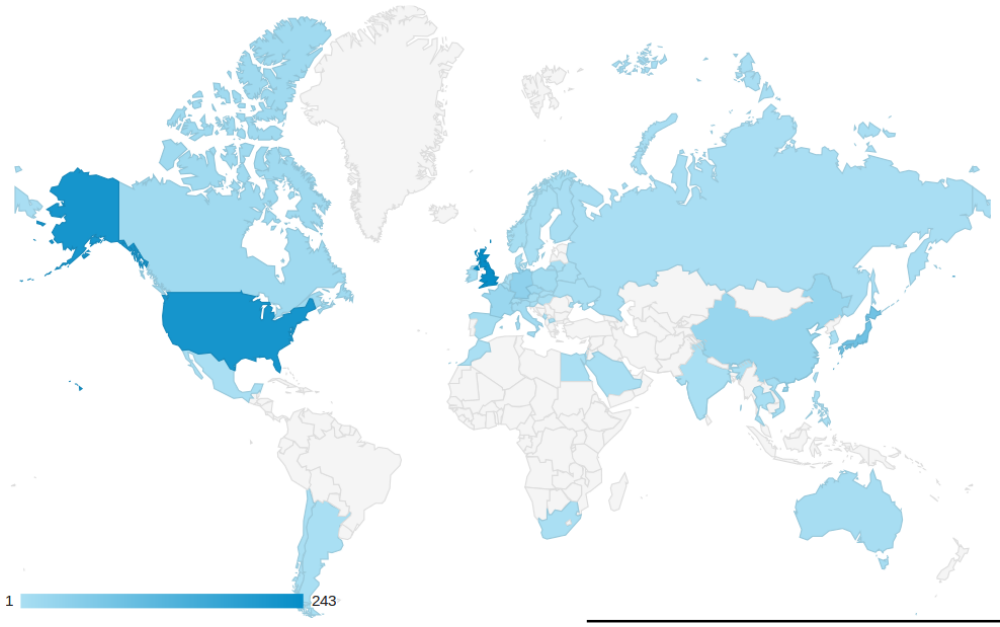
fliang: Collins only uses expert evaluators

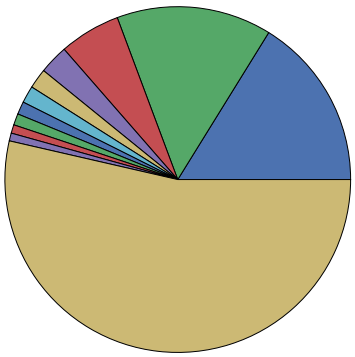
In comparison, a recently published comparable system for generating Bach-styled music found that 20% of its evaluators were able to discriminate significantly better than chance.

fig. 7.6 also shows that participants had more trouble discriminating entire compositions (SATB) than harmonizations (AT, ATB) where a subset of the parts have already been given. While this may seem counterintuitive, recall that the model in

fliang: reference

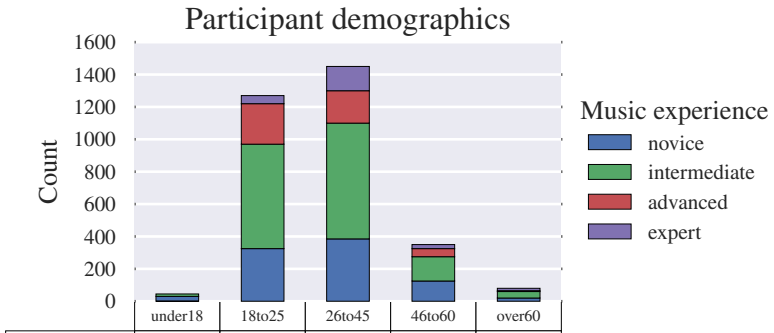
is uni-directional and does not account for any future constraints on other parts. We made this design decision intentionally because one of our requirements was sampling the model for novel compositions. However, since harmonization tasks provide the full past and future context for other parts, they effectively impose constraints on LSTM hidden state dynamics. We





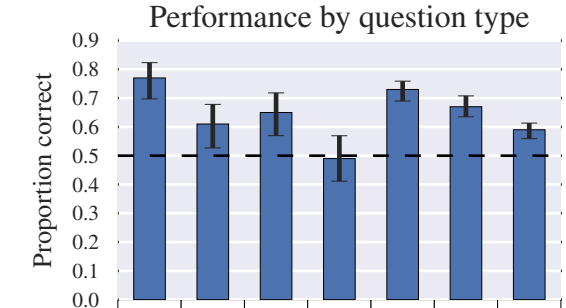
Country	Responses	Proportion
UK	243	16.0%
US	218	15.0%
Japan	86	6.0%
Germany	41	3.0%
China	28	2.0%
France	24	2.0%
Czechia	18	1.0%
Canada	16	1.0%
Italy	12	1.0%
Poland	11	1.0%
Other	805	54.0%

Fig. 7.4 Geographic distribution of participants



	under18	18to25	26to45	46to60	over60
novice	30.0	325.0	385.0	125.0	20.0
intermediate	15.0	645.0	715.0	150.0	40.0
advanced	0.0	250.0	200.0	50.0	5.0
expert	0.0	50.0	150.0	25.0	15.0

Fig. 7.5 Demographics of participants



	S	A	T	B	AT	ATB	SATB
Proportion	0.77	0.61	0.65	0.49	0.73	0.67	0.59

Fig. 7.6 responses-Mask

expect methods which account for both future and past context (*e.g.* using the output sequence from a bidirectional RNN

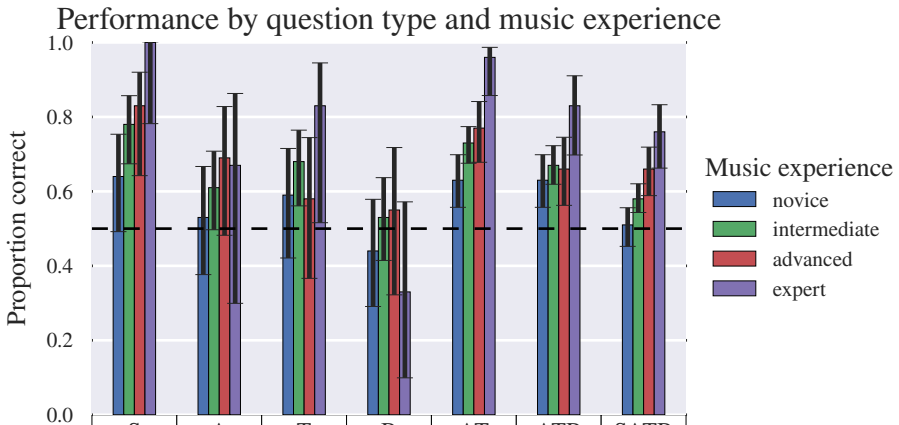
fliang: cite

inputs) to mitigate this problem, which we leave for future work.

When only a single part is composed by BachBot, we find the results vary significantly across different parts. Composing the soprano part proved to be the easiest to discriminate, an unsurprising result given that in chorale style music soprano parts are responsible for the melody

fliang: cite

. Composing the alto and tenor parts achieved similar performance as composing all four parts, a result which may also be caused by not accounting for future constraints on model



	S	A	T	B	AT	ATB	SATB
novice	0.64	0.53	0.59	0.44	0.63	0.63	0.51
intermediate	0.78	0.61	0.68	0.53	0.73	0.67	0.58
advanced	0.83	0.69	0.58	0.55	0.77	0.66	0.66
expert	1.0	0.67	0.83	0.33	0.96	0.83	0.76

Fig. 7.7 Proportion of correct responses for each question type and music experience level.

outputs. Removing the bass proved to be the most perceptually difficult to discern from real Bach.

In fig. 7.7, responses are further segmented by music experience. Unsurprisingly, we find that the proportion of correct responses correlates positively with experience.

fliang: Comment more about these figures in captions

fliang: Mark: Surprising that that new bass is so successful. Worthy of discussion.

fig. 7.8 shows the proportion correct for each question. Encouragingly, it shows that 41.7%

fliang: VERIFY LAST

of the SATB pairs were not statistically different than baseline, suggesting that **while not always consistent BachBot is capable of composing music which the average participant cannot discern from actual Bach**.

fliang: Reference Future work: analyze bad examples in fig. 7.8

7.3 User feedback

In this section, we showcase some of the particularly helpful feedback we received from our users and observe that some common themes are present among the features participants used to discriminate synthetic from original Bach.

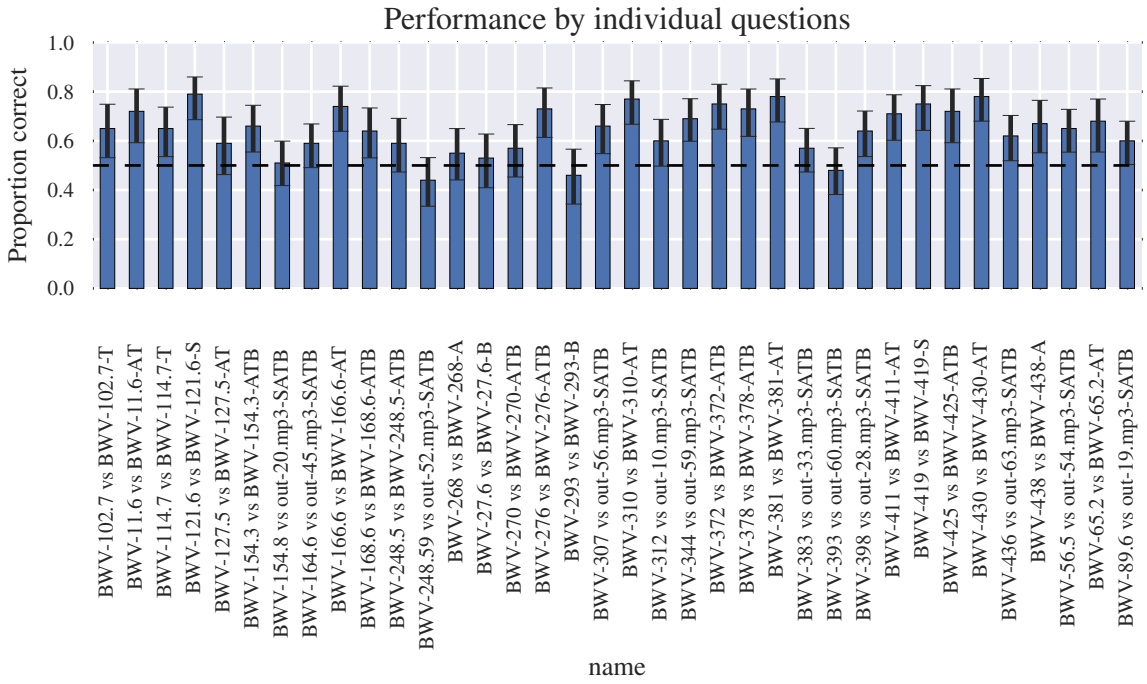


Fig. 7.8 Proportion of correct responses broken down by individual questions.

Most critically, we found that lack of convincing phrasing to be one of the most cited difference between BachBot and real Back:

> The modulations and part writing were the giveaway for me (and once or twice the phrasing)

> Got 5/5. The trick is to listen for the unnatural pauses at regular intervals.

> Cool project, I scored 100% so I'm quite pleased with myself ;o) I do play an instrument although I'm not classical trained. If I had an inkling to why I could choose the background phrasing of the Bach pieces is far more elegant than the computer generated pieces.

> really impressive! If I didn't know about counterpoint that quiz would've stumped me

We also found that certain participants who have studied Bach chorales could recognize certain scores despite normalizing key and tempo:

> [BWV-248.5 ATB] #2: there was a first inversion at a strong cadence point that just wasn't credible in the non-Bach clip (of course I am sure you can point to a piece of Bach that does end on a first inversion!)

[BWV-248.5 ATB] #3: this was the hardest one to judge in my opinion. I went on gut feeling of the first section, before the very slow part that I suspect may be an artefact of the tempo normalisation that you have done. Were there perhaps fewer open chords in the genuine Bach?

[BWV-11.6 B] #4: the harmony felt quit wrong in this one - too many weird chromatic harmonies and semitone clashes even for JSB and it ended in the wrong key.

[BWV-436 SATB] #5: this one I found quite easy, partly because I recognised it, but also because the algorithm was perhaps trying to be too clever and turn too many harmonic corners in too quick a succession

Further, although I couldn't quote BWV numbers to you, I recognised samples 2 and 5, though the performance of #2 in particular seemed highly ornamented and embellished.

The whole thing is really quite impressive overall though…

7.4 Competitive analysis of large-scale evaluation methodologies

fliang: Breakdown costs of Azure CDN, App Service, BlobStore. Most expensive was domain registration

fliang: Compare costs and quality with MTurk

Higher quality. Music experts are not usually doing tasks on MTurk, but would be very interested in an open-source research project. In contrast, 20%

fliang: UPDATE THIS NUMBER

of our participants have either formally studied or taught music theory.

Payments on MTurk are suggested to follow a reasonable hourly rate, with an example of $8 per hour or about 13c per minute. In practice, many mTurk tasks pay much less overall, with the median study paying just 5-10c for a task taking "a few minutes," like watching and providing feedback on 3 short (15-second) videos, summarizing a website, and evaluating hypothetical and real market products. Indeed, "wages" this low have been shown to result in lower quality output than could be had for no payment at all, by pure volunteers.

[40]

MTurk is also crestricted to US participants only[108].

Paid service providers cost anywhere from $20 to $55 per month just for authoring tools and server space[142] At the time of writing, paid responses cost $1.50–$3.00 on SurveyMonkey [102].

What can we say about the perception of music by the silent majority of listeners, those for whom music is written but who neither create music nor can articulate their musical experience? How do they acquire their demonstrably sophisticated intuitions about music patterns typical of their culture? Experiments in the cognitive psychology of music have cast some light on the first question. Recent developments in neural net learning now enables us to explore the second.

Bharucha and Todd [14]

8 Discussion, Conclusions, and Future Work

8.1 Discussion

8.1.1 Contributions

- An encoding scheme for representing music with arbitrary degrees of polyphony as ordered sequences of tokens
- Brought together recent methods from deep learning to develop a sequence prediction model which avoids any hand-crafted input features and minimizes domain-specific design choices
- Optimized the performance of the proposed model and quantitatively evaluated its performance on both composition and harmonization tasks
- Performed the largest (as of Thursday 11th August, 2016) published musical Turing test of an automatic composition system
- Investigated the internal representations learned by the model, identifying neurons specific to music-theoretic concepts.

8.2 Conclusions

Recall that our journey was prompted by the questions: can the current state-of-the-art in deep learning be used to build a pure connectionist model which learns to compose in a style indistinguishable from Bach. To answer this question, we took numerous ideas from deep learning research and built a deep LSTM language model for Bach.

While our automatic composition model does surprisingly well in composition tasks, it underperforms when asked to harmonize fixed melodies: a task which music theorists consider significantly easier than automatic composition.

Both the models the and large-scale evaluation frameworks have been open sourced. The models are currently being ported to run on Tensorflow and will be part of the Google Magenta project. We plan to continue running the evaluation study on bachbot.com and will update the website as more responses are collected. A press release on SoftwareEngineeringDaily (>10K daily users) and a feature on Cambridge University's front page are both scheduled to go live early September, so we expect a significant increase in the amount of responses.

8.3 Extensions and Future Work

Looking through fig. 7.8 on page 81, we see that certain samples are virtually indistinguishable from Bach while others can be identified more than 80% of the time. It would be valuable to understand what features are present in the failure cases which make them easy to distinguish, a potential extension for any interested music theorists.

One significant opportunity for improvement is to account for future context during harmonization tasks. Specifically, the requirement that our model can be sampled to generate compositions constrains its architecture to be unidirectional, significantly impacting its performance on harmonization tasks where future outputs are constrained.

One method to address this is to apply bidirectional RNNs[63] and the sequence to sequence framework[126] to map the constrained parts to the harmonized parts while accounting for both past and future context. An attention mechanism similar to Bahdanau, Cho, and Bengio [7] could be introduced on top of the bidirectional RNN to both enable the model to selectively attend to specific time intervals within the context as well as provide insight into what the model deems relevant when generating harmonies.

Another way to account for future constraints is using a lookahead search. Instead of generating outputs by greedily sampling the RNN's predictions at each time, the RNN is expanded for multiple timesteps and the overall best path is selected. This approach is significantly complicated by an exponential growth in possible states ($O(128^L)$ states when looking ahead L

timesteps), but nevertheless can be made computationally tractable using approximation techniques like beam search[101].

Another interesting area for further exploration is in how the different parts are ordered when flattening music scores (where all the notes in a chord are played simultaneously) into encoded token sequences (where a sequential ordering is imposed on the notes in a chord). Harmonization results from section 6.4 on page 70 showed that the Soprano and Alto parts achieved significantly higher error rates than Tenor and Bass parts, which might be attributed the SATB order imposed when encoding chords. We expect ordering the parts to be harmonized last should improve the model as the fixed parts can now provide additional context aiding more consistent harmony prediction.

We use the Bach chorales because they are fairly homogeneous, widely available in a respectable quantity, and well studied by music theorists. However, our proposed encoding scheme extends to arbitrary degrees of polyphony and musical style. As evidenced by fig. 6.2 on page 71, the learned model is already able to plausibly harmonize melodies which differ significantly from Bach's baroque style. An interesting extension would be to further investigate the limits of of our model's generalality and its failure modes by applying the model to other styles of music.

In the criticism of musical Turing tests by Ariza [5], one of the major points of concern is the difficulty of leveraging feedback to improve the system. Instead, they recommend conducting listening tests and collecting subjective feedback in natural language as done in Collins et al. [22]. One direction for future work would be to act on their recommendations by conducting listening tests and analyzing the responses to identify and prioritize areas for improvement.

References

[1] **abc:standard [abc wiki]**. July 2015. URL: http://abcnotation.com/wiki/abc:standard.

[2] Moray Allan and Christopher KI Williams. "allan2005". In: **Advances in Neural Information Processing Systems** 17 (2005), pp. 25–32.

[3] Adam Alpern. "Techniques for algorithmic composition of music". In: **On the web: http://hamp. hampshire. edu/adaF92/algocomp/algocomp** 95 (1995), p. 120.

[4] Charles Ames. "The Markov process as a compositional model: A survey and tutorial". In: **Leonardo** (1989), pp. 175–187.

[5] Christopher Ariza. "The interrogator as critic: The turing test and the evaluation of generative music systems". In: **Computer Music Journal** 33.2 (2009), pp. 48–70.

[6] Christopher Ariza and M Cuthbert. **Modeling beats, accents, beams, and time signatures hierarchically with music21 meter objects**. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 2010.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "#4 Neural Machine Translation by Jointly Learning to Align and Translate". In: (2015), pp. 1–15. arXiv: 1409.0473. URL: http://arxiv.org/pdf/1409.0473v6.pdf.

[8] Justin Bayer et al. "On fast dropout and its applicability to recurrent networks". In: **arXiv preprint arXiv:1311.0701** (2013).

[9] Matthew I Bellgard and Chi-Ping Tsang. "Harmonizing music the Boltzmann way". In: **Connection Science** 6.2-3 (1994), pp. 281–297.

[10] Samy Bengio et al. "Scheduled sampling for sequence prediction with recurrent neural networks". In: **Advances in Neural Information Processing Systems**. 2015, pp. 1171–1179.

[11] Yoshua Bengio. "Learning deep architectures for AI". In: **Foundations and trends® in Machine Learning** 2.1 (2009), pp. 1–127.

[12] Yoshua Bengio and Olivier Delalleau. "On the expressive power of deep architectures". In: **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)** 6925 LNAI (2011), pp. 18–36. ISSN: 03029743. DOI: 10.1007/978-3-642-24412-4_3. arXiv: 1206.5533.

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning Long-Term Dependencies with Gradient Descent is Difficult". In: **IEEE Transactions on Neural Networks** 5.2 (1994), pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181. arXiv: arXiv:1211.5063v2. URL: http://jmlr.org/proceedings/papers/v28/pascanu13.pdf.

[14] Jamshed J Bharucha and Peter M Todd. "Modeling the perception of tonal structure with neural nets". In: **Computer Music Journal** 13.4 (1989), pp. 44–53.

[15] Nicolas Boulanger-Lewandowski, Pascal Vincent, and Yoshua Bengio. "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription". In: **Proceedings of the 29th International Conference on Machine Learning (ICML-12)** Cd (2012), pp. 1159–1166. arXiv: 1206.6392.

[16] Edmund Bowles. "Musicke's handmaiden: Or technology in the service of the arts". In: **The computer and music** (1970), pp. 3–20.

[17] Tim O Brien and Iran Roman. "A Recurrent Neural Network for Musical Structure Processing and Expectation". In: **CS224d: Deep Learning for Natural Language Processing Final Projects** (2016), pp. 1–9.

[18] Arthur E Bryson, Walter F Denham, and Stewart E Dreyfus. "Optimal programming problems with inequality constraints". In: **AIAA journal** 1.11 (1963), pp. 2544–2550.

[19] John Butt. "Bach-Werke-Verzeichnis". In: **Notes** 55.4 (1999), pp. 890–893.

[20] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: **arXiv preprint arXiv:1406.1078** (2014).

[21] Ching-Hua Chuan and Elaine Chew. "A hybrid system for automatic generation of style-specific accompaniment". In: **Proceedings of the 4th International Joint Workshop on Computational Creativity**. 2007, pp. 57–64.

[22] Tom Collins et al. "Developing and evaluating computational models of musical style". In: **Artificial Intelligence for Engineering Design, Analysis and Manufacturing** 30.01 (2016), pp. 16–43.

[23] Wikimedia Commons. **Comparison of various duple note values.** File:Duple note values comparison.png. 2012. URL: https://en.wikipedia.org/wiki/File:Duple_note_values_comparison.png (visited on 08/10/2016).

[24] Darrell Conklin and Ian H Witten. "Multiple viewpoint systems for music prediction". In: **Journal of New Music Research** 24.1 (1995), pp. 51–73.

[25] Grosvenor Cooper and Leonard B Meyer. **The rhythmic structure of music**. Vol. 118. University of Chicago Press, 1963.

[26] David Cope. "Computer modeling of musical intelligence in EMI". In: **Computer Music Journal** 16.2 (1992), pp. 69–83.

[27] David Cope. "Experiments in Music Intelligence". In: **Proceedings of the International Computer Music Conference**. 1987.

[28] David Cope. "The well-programmed clavier: Style in computer music composition". In: **XRDS: Crossroads, The ACM Magazine for Students** 19.4 (2013), pp. 16–20.

[29] David H Cope. **Recombinant music composition algorithm and method of using the same**. US Patent 7,696,426. Apr. 2010.

[30] Eduardo Coutinho et al. "Computational musicology: An artificial life approach". In: **2005 portuguese conference on artificial intelligence**. IEEE. 2005, pp. 85–93.

[31] Pedro P Cruz-Alcázar and Enrique Vidal-Ruiz. "Learning regular grammars to model musical style: Comparing different coding schemes". In: **International Colloquium on Grammatical Inference**. Springer. 1998, pp. 211–222.

[32] Michael Scott Cuthbert and Christopher Ariza. "music21: A toolkit for computer-aided musicology and symbolic music data". In: (2010).

[33] Michael Scott Cuthbert et al. “Hidden Beyond MIDI’s Reach : Feature Extraction and Machine Learning with Rich Symbolic Formats in music21”. In: **Proceedings of the Neural Information Processing Systems Conference** (2011), pp. 3–4.

[34] George Cybenko. “Degree of approximation by superpositions of a sigmoidal function”. In: **Approximation Theory and its Applications** 9.3 (1993), pp. 17–28. ISSN: 10009221. DOI: 10.1007/BF02836480.

[35] Roger B Dannenberg, Belinda Thom, and David Watson. “A machine learning approach to musical style recognition”. In: (1997).

[36] James Denny. **The Oxford school harmony course**. Vol. 1. Oxford University Press, 1960.

[37] Christine Denton and M Fillion. “The History of Musical Tuning and Temperament during the Classical and Romantic Periods”. In: (1997).

[38] Jacob Devlin et al. “Fast and Robust Neural Network Joint Models for Statistical Machine Translation.” In: **ACL (1)**. Citeseer. 2014, pp. 1370–1380.

[39] Mark Dolson. “Machine tongues XII: Neural networks”. In: **Computer Music Journal** 13.3 (1989), pp. 28–40.

[40] Julie S Downs et al. “Are your participants gaming the system?: screening mechanical turk workers”. In: **Proceedings of the SIGCHI Conference on Human Factors in Computing Systems**. ACM. 2010, pp. 2399–2402.

[41] Kemal Ebcioğlu. “An expert system for harmonizing four-part chorales”. In: **Computer Music Journal** 12.3 (1988), pp. 43–51.

[42] D. Eck and J. Schmidhuber. “Finding temporal structure in music: Blues improvisation with LSTM recurrent networks”. In: **Neural Networks for Signal Processing - Proceedings of the IEEE Workshop** 2002-Janua (2002), pp. 747–756. ISSN: 0780376161. DOI: 10.1109/NNSP.2002.1030094.

[43] Douglas Eck and Jasmin Lapalme. “Learning musical structure directly from sequences of music”. In: **University of Montreal, Department of Computer Science, CP** 6128 (2008).

[44] Douglas Eck and Jürgen Schmidhuber. “A First Look at Music Composition using LSTM Recurrent Neural Networks”. In: **Idsia** (2002). URL: http://www.idsia.ch/%7B~%7Djuergen/blues/IDSIA-07-02.pdf.

[45] Salah El Hihi and Yoshua Bengio. “Hierarchical Recurrent Neural Networks for Long-Term Dependencies.” In: **NIPS**. Vol. 400. Citeseer. 1995, p. 409.

[46] Jeffrey L Elman. “Finding structure in time”. In: **Cognitive science** 14.2 (1990), pp. 179–211.

[47] Johannes Feulner and Dominik Hörnel. “Melonet: Neural networks that learn harmony-based melodic variations”. In: **Proceedings of the International Computer Music Conference**. INTERNATIONAL COMPUTER MUSIC ACCOCIATION. 1994, pp. 121–121.

[48] JA Franklin et al. “Learning and improvisation”. In: **Neural Information Processing Systems**. Vol. 14. 2001.

[49] Judy A Franklin. “Jazz Melody Generation from Recurrent Network Learning of Several Human Melodies.” In: **FLAIRS Conference**. 2005, pp. 57–62.

[50] Judy A Franklin. "Predicting reinforcement of pitch sequences via lstm and td". In: **Proc. of International Computer Music Conference, Miami, Florida**. Vol. 306. 2004.

[51] Judy A Franklin. "Recurrent Neural Networks and Pitch Representations for Music Tasks." In: **FLAIRS Conference**. 2004, pp. 33–37.

[52] Judy A Franklin. "Recurrent neural networks for music computation". In: **INFORMS Journal on Computing** 18.3 (2006), pp. 321–338.

[53] Dylan Freedman. "Correlational Harmonic Metrics: Bridging Computational and Human Notions of Musical Harmony". PhD thesis. 2015.

[54] Felix A Gers and E Schmidhuber. "LSTM recurrent networks learn simple context-free and context-sensitive languages". In: **IEEE Transactions on Neural Networks** 12.6 (2001), pp. 1333–1340.

[55] Felix A Gers and Jürgen Schmidhuber. "Recurrent nets that time and count". In: **Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on**. Vol. 3. IEEE. 2000, pp. 189–194.

[56] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM". In: **Neural computation** 12.10 (2000), pp. 2451–2471.

[57] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. "Learning precise timing with LSTM recurrent networks". In: **Journal of machine learning research** 3.Aug (2002), pp. 115–143.

[58] Felix A Gers et al. "DEKF-LSTM." In: **ESANN**. 2002, pp. 369–376.

[59] Kratarth Goel, Raunaq Vohra, and JK Sahoo. "Polyphonic music generation by modeling temporal dependencies using a RNN-DBN". In: **International Conference on Artificial Neural Networks**. Springer. 2014, pp. 217–224.

[60] Christoph Goller and Andreas Kuchler. "Learning task-dependent distributed representations by backpropagation through structure". In: **Neural Networks, 1996., IEEE International Conference on**. Vol. 1. IEEE. 1996, pp. 347–352.

[61] Ian Goodfellow et al. "Generative adversarial nets". In: **Advances in Neural Information Processing Systems**. 2014, pp. 2672–2680.

[62] Alex Graves. "Generating sequences with recurrent neural networks". In: **arXiv preprint arXiv:1308.0850** (2013).

[63] Alex Graves and J??rgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM networks". In: **Proceedings of the International Joint Conference on Neural Networks** 4 (2005), pp. 2047–2052. ISSN: 08936080. DOI: 10.1109/IJCNN.2005.1556215.

[64] Klaus Greff et al. "LSTM: A search space odyssey". In: **arXiv preprint arXiv:1503.04069** (2015).

[65] Niall Griffith and Peter M Todd. **Musical networks: Parallel distributed perception and performance**. MIT Press, 1999.

[66] PDP Research Group et al. "Parallel distributed processing: Explorations in the microstructure of cognition: Vol. 1". In: **Foundations. Cambridge, MA: MIT Press** (1986).

[67] Stephen Handel. **Listening: An introduction to the perception of auditory events.** The MIT Press, 1993.

[68] Kenneth Heafield et al. "Scalable Modified Kneser-Ney Language Model Estimation". In: **Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics**. Sofia, Bulgaria, Aug. 2013. URL: http://kheafield.com/professional/edinburgh/estimate%5C_paper.pdf.

[69] William Herlands et al. "A Machine Learning Approach to Musically Meaningful Homogeneous Style Classification". In: **Twenty-Eighth AAAI Conference on Artificial Intelligence** (2014), pp. 276–282. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8314.

[70] Hermann Hild, Johannes Feulner, and Wolfram Menzel. "HARMONET: A neural net for harmonizing chorales in the style of JS Bach". In: **NIPS**. 1991, pp. 267–274.

[71] Geoffrey E Hinton and Terrence J Sejnowski. "Learning and releaming in Boltzmann machines". In: **Parallel distributed processing: Explorations in the microstructure of cognition** 1 (1986), pp. 282–317.

[72] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: **Neural computation** 9.8 (1997), pp. 1735–1780.

[73] Peter Hoffmann. "Towards an" automated art": Algorithmic processes in Xenakis' compositions". In: **Contemporary Music Review** 21.2-3 (2002), pp. 121–131.

[74] Dominik Hörnel. "MELONET I: Neural nets for inventing baroque-style chorale variations". In: **NIPS**. 1997, pp. 887–893.

[75] Dominik Hornel and Tomas Ragg. "Learning Musical Structure and Style by Recognition, Prediction and Evolution". In: (1996).

[76] Brian Hyer. **Tonality**. URL: http://www.oxfordmusiconline.com/subscriber/article/grove/music/28102 (visited on 08/10/2016).

[77] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: **arXiv preprint arXiv:1502.03167** (2015).

[78] Oswald Jonas and John Rothgeb. **Introduction to the theory of Heinrich Schenker: the nature of the musical work of art**. New York: Longman, 1982.

[79] Michael I Jordan. "Serial order: A parallel distributed processing approach". In: **Advances in psychology** 121 (1997), pp. 471–495.

[80] Łukasz Kaiser and Ilya Sutskever. "Neural gpus learn algorithms". In: **arXiv preprint arXiv:1511.08228** (2015).

[81] Barry Kernfeld. **Meter**. URL: www.oxfordmusiconline.com/subscriber/article_citations/grove/music/J298700 (visited on 08/10/2016).

[82] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: **arXiv preprint arXiv:1412.6980** (2014).

[83] Jan Koutnik et al. "A Clockwork RNN". In: **Proceedings of The 31st International Conference on Machine Learning** 32 (2014), pp. 1863–1871. arXiv: arXiv:1402.3511v1. URL: http://jmlr.org/proceedings/papers/v32/koutnik14.html.

[84] Carol L Krumhansl. **Cognitive foundations of musical pitch**. Oxford University Press, 2001.

[85] Bernice Laden and Douglas H Keefe. "The representation of pitch in a neural net model of chord classification". In: **Computer Music Journal** 13.4 (1989), pp. 12–26.

[86] Fred Lerdahl. **Jackendoff. A Generative Theory of Tonal Music**. 1983.

[87] Rensis Likert. "A technique for the measurement of attitudes." In: **Archives of psychology** (1932).

[88] Seppo Linnainmaa. "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors". In: **Master's Thesis (in Finnish), Univ. Helsinki** (1970), pp. 6–7.

[89] I-Ting Liu and Bhiksha Ramakrishnan. "Bach in 2014: Music Composition with Recurrent Neural Network". In: **arXiv:1412.3191** 5 (2014), pp. 1–9. arXiv: 1412.3191. URL: http://arxiv.org/abs/1412.3191.

[90] Xunying Liu et al. "Efficient lattice rescoring using recurrent neural network language models". In: **2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. IEEE. 2014, pp. 4908–4912.

[91] Qi Lyu. "Polyphonic Music Modelling with LSTM-RTRBM". In: **Proceedings of the 23rd Annual ACM Conference on Multimedia Conference** (2015), pp. 991–994.

[92] Michael I Mandel, Graham E Poliner, and Daniel PW Ellis. "Support vector machine active learning for music retrieval". In: **Multimedia systems** 12.1 (2006), pp. 3–13.

[93] Tomáš Mikolov. "Statistical Language Models Based on Neural Networks". PhD thesis. Brno, CZ, 2012, p. 129. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10158.

[94] Tomas Mikolov et al. "Learning Longer Memory in Recurrent Neural Networks". In: **Iclr** (2015), pp. 1–9. arXiv: arXiv:1412.7753v1. URL: http://arxiv.org/pdf/1412.7753v1.pdf.

[95] T Mikolov et al. "Recurrent Neural Network based Language Model". In: **Interspeech** September (2010), pp. 1045–1048.

[96] Michael C Mozer. "Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing". In: **Connection Science** 6.2-3 (1994), pp. 247–280.

[97] Dylan Jeremy Nagler. "SCHUBOT: Machine Learning Tools for the Automated Analysis of Schubert's Lieder". PhD thesis. 2014.

[98] Jean-Jacques Nattiez. **Music and discourse: Toward a semiology of music**. Princeton University Press, 1990.

[99] Aran Nayebi and Matt Vitelli. "GRUV: Algorithmic Music Generation using Recurrent Neural Networks". In: **CS224d: Deep Learning for Natural Language Processing Final Projects** (2015), pp. 1–6.

[100] Yizhao Ni et al. "An end-to-end machine learning system for harmonic analysis of music". In: **IEEE Transactions on Audio, Speech, and Language Processing** 20.6 (2012), pp. 1771–1783.

[101] Peter Norvig. **Paradigms of artificial intelligence programming: case studies in Common LISP**. Morgan Kaufmann, 1992.

[102] **Online Research Panel Pricing | SurveyMonkey Audience**. URL: https://www.surveymonkey.co.uk/mp/audience/audience-pricing/.

[103] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: **Proceedings of The 30th International Conference on Machine Learning** 2 (2012), pp. 1310–1318. ISSN: 1045-9227. DOI: 10.1109/72.279181. arXiv: arXiv:1211.5063v2. URL: http://jmlr.org/proceedings/papers/v28/pascanu13.pdf.

[104] Razvan Pascanu et al. "How to construct deep recurrent neural networks". In: **arXiv preprint arXiv:1312.6026** (2013).

[105] Marcus Pearce, David Meredith, and Geraint Wiggins. "Motivations and methodologies for automation of the compositional process". In: **Musicae Scientiae** 6.2 (2002), pp. 119–147.

[106] Marcus Pearce and Geraint Wiggins. "Towards a framework for the evaluation of machine compositions". In: **Proceedings of the AISB'01 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences**. Citeseer. 2001, pp. 22–32.

[107] Walter Piston. "Harmony. (Revised and expanded by Mark DeVoto)". In: **Londres: Victor Gollancz LTD** (1978).

[108] Donya Quick. "Kulitta: A Framework for Automated Music Composition". PhD thesis. YALE UNIVERSITY, 2014.

[109] Donya Quick and Paul Hudak. **A temporal generative graph grammar for harmonic and metrical structure**. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 2013.

[110] Daniel Ramage. "Hidden Markov models fundamentals". In: **Lecture Notes. http://cs229.stanford. edu/section/cs229-hmm. pdf** (2007).

[111] Don Michael Randel. **The Harvard concise dictionary of music and musicians**. Harvard University Press, 1999.

[112] Martin Riedmiller and Heinrich Braun. "A direct adaptive method for faster backpropagation learning: The RPROP algorithm". In: **Neural Networks, 1993., IEEE International Conference On**. IEEE. 1993, pp. 586–591.

[113] AJ Robinson and Frank Fallside. **The utility driven dynamic error propagation network**. University of Cambridge Department of Engineering, 1987.

[114] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: **Cognitive modeling** 5.3 (1988), p. 1.

[115] Heinrich Schenker. "Harmony, ed. Oswald Jonas, trans. Elisabeth Mann Borgese". In: **Chicago: University of Chicago Press** 35 (1954), pp. 105–32.

[116] Jürgen Schmidhuber. "Learning complex, extended sequences using the principle of history compression". In: **Neural Computation** 4.2 (1992), pp. 234–242.

[117] Marco Scirea et al. "MetaCompose: A Compositional Evolutionary Music Composer". In: **International Conference on Evolutionary and Biologically Inspired Music and Art**. Springer. 2016, pp. 202–217.

[118] Roger N Shepard. “Geometrical approximations to the structure of musical pitch.” In: **Psychological review** 89.4 (1982), p. 305.

[119] Randall R Spangler, Rodney M Goodman, and Jim Hawkins. “Bach in a Box-Real-Time Harmony”. In: (1998).

[120] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting.” In: **Journal of Machine Learning Research** 15.1 (2014), pp. 1929–1958.

[121] Efstathios Stamatatos and Gerhard Widmer. “Automatic identification of music performers with learning ensembles”. In: **Artificial Intelligence** 165.1 (2005), pp. 37–56.

[122] Andreas Stolcke et al. “SRILM-an extensible language modeling toolkit.” In: **Interspeech**. Vol. 2002. 2002, p. 2002.

[123] Bob L Sturm et al. “Music transcription modelling and composition using deep learning”. In: **arXiv preprint arXiv:1604.08723** (2016).

[124] Bob Sturm, Joao Felipe Santos, and Iryna Korshunova. “Folk music style modelling by recurrent neural networks with long short term memory units”. In: **16th International Society for Music Information Retrieval Conference**. 2015.

[125] Ilya Sutskever. “Training Recurrent Neural Networks - Ilia Sutskever - PhD thesis”. In: (). URL: http://www.cs.utoronto.ca/%5C%7Eilya/pubs/ilya%5C_sutskever%5C_phd%5C_thesis.pdf.

[126] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to sequence learning with neural networks”. In: **Advances in neural information processing systems**. 2014, pp. 3104–3112.

[127] Richard S Sutton and Andrew G Barto. **Reinforcement learning: An introduction**. Vol. 1. 1. MIT press Cambridge, 1998.

[128] Ernst Terhardt. “Pitch, consonance, and harmony”. In: **The Journal of the Acoustical Society of America** 55.5 (1974), pp. 1061–1069. DOI: http://dx.doi.org/10.1121/1.1914648. URL: http://scitation.aip.org/content/asa/journal/jasa/55/5/10.1121/1.1914648.

[129] Peter Todd. “A sequential network design for musical applications”. In: **Proceedings of the 1988 connectionist models summer school**. 1988, pp. 76–84.

[130] Peter M Todd. “A connectionist approach to algorithmic composition”. In: **Computer Music Journal** 13.4 (1989), pp. 27–43.

[131] Petri Toiviainen. **Symbolic AI versus Connectionism in Music Research.** 2000.

[132] Chi Ping Tsang and Melanie Aitken. “Harmonizing Music as a Discipline in Contraint Logic Programming”. In: **Proceedings of the International Computer Music Conference**. INTERNATIONAL COMPUTER MUSIC ACCOCIATION. 1991, pp. 61–61.

[133] Alan M Turing. “Computing machinery and intelligence”. In: **Mind** 59.236 (1950), pp. 433–460.

[134] Dmitri Tymoczko. “Three conceptions of musical distance”. In: **Communications in Computer and Information Science** 38 (2009), pp. 258–272. ISSN: 18650929. DOI: 10.1007/978-3-642-02394-1_24.

[135] Eric Wanner. "The ATN and the sausage machine: Which one is baloney?" In: **Cognition** 8.2 (1980), pp. 209–225.

[136] Gil Weinberg et al. "A real-time genetic algorithm in human-robot musical improvisation". In: **International Symposium on Computer Music Modeling and Retrieval**. Springer. 2007, pp. 351–359.

[137] John David White and William E Lake. **Guidelines for college teaching of music theory**. Scarecrow Press, 2002.

[138] Ronald J Williams and Jing Peng. "An efficient gradient-based algorithm for on-line training of recurrent network trajectories". In: **Neural computation** 2.4 (1990), pp. 490–501.

[139] Ronald J Williams and David Zipser. "A learning algorithm for continually running fully recurrent neural networks". In: **Neural computation** 1.2 (1989), pp. 270–280.

[140] Ronald J Williams and David Zipser. "Gradient-based learning algorithms for recurrent networks and their computational complexity". In: **Back-propagation: Theory, architectures and applications** (1995), pp. 433–486.

[141] Terry Winograd. "Linguistics and the computer analysis of tonal harmony". In: **journal of Music Theory** 12.1 (1968), pp. 2–49.

[142] Kevin B. Wright. "Researching Internet-Based Populations: Advantages and Disadvantages of Online Survey Research, Online Questionnaire Authoring Software Packages, and Web Survey Services". In: **Journal of Computer-Mediated Communication** 10.3 (2005), pp. 00–00. ISSN: 1083-6101. DOI: 10.1111/j.1083-6101.2005.tb00259.x. URL: http://dx.doi.org/10.1111/j.1083-6101.2005.tb00259.x.

[143] Wojciech Zaremba. "An empirical exploration of recurrent network architectures". In: (2015).

[144] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. "Recurrent neural network regularization". In: **arXiv preprint arXiv:1409.2329** (2014).

A

Appendix A: A primer on Western music theory

This chapter serves as a primer on the music theory knowledge assumed throughout the rest of our work. It synthesizes material originally presented in Franklin [52], Nagler [97], Quick [108], and Freedman [53]. Readers with a strong music background may wish to skip this section.

Music theory is a branch of musicology concerned with the study of the rules and practices of music. While the general field includes study of acoustic qualities such as dynamics and timbre, we restrict the scope of our research to modeling musical **scores** (*e.g.* fig. A.1) and neglect issues related to articulation and performance (*e.g.* dynamics, accents, changes in tempo) as well as synthesis/generation of the physical acoustic waveforms.

This is justified because the physical waveforms are more closely related to the skill of the performers and instruments used and are likely to vary significantly across different performances. Furthermore, articulations in the same musical piece may differ across transcriptions and performers. Despite these variations, a piece of music is still recognizable from just the notes, suggesting that notes are the defining element for a piece of music.

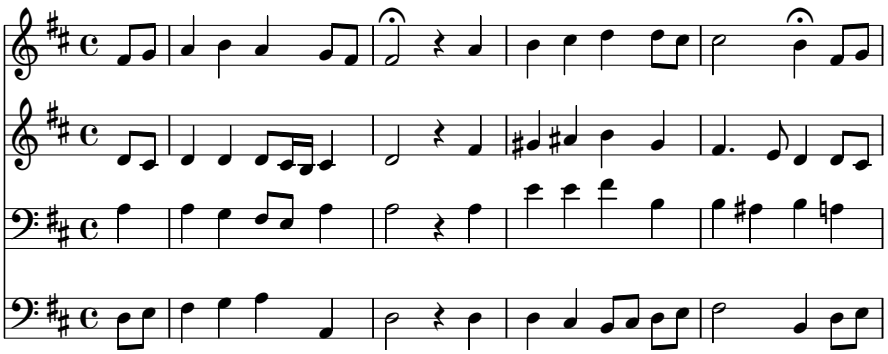


Fig. A.1 Sheet music representation of the first four bars of BWV 133.6

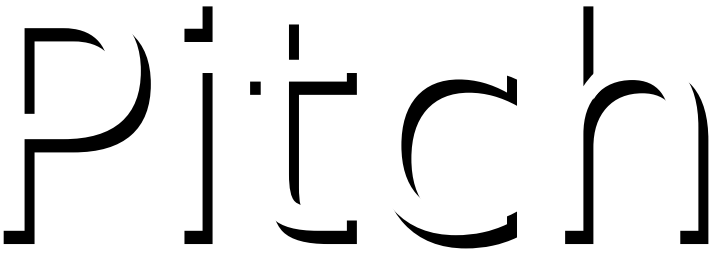


Fig. A.2 Terhardt's visual analogy for pitch. Similar to how the viewer of this figure may percieve contours not present, pitch describes subjective information received by the listener even when physical frequencies are absent.

A.1 Notes: the basic building blocks

A **note** is the most fundamental element of music score and represents a sound played at a certain **pitch** for a certain **duration**. In sheet music such as fig. A.1, the notes are denoted by the filled/unfilled black heads with protruding stems. As a can be viewed as a collection of notes over time, notes are the fundamental building blocks for musical scores.

A.1.1 Pitch

Though pitch is closely related to physical frequency of vibration of a waveform (as measured in Hertz), pitch is a perceptual property whose semantic meaning is derived from a listener's perception. This distinction has been scrutinized by Terhardt [128], whose visual analogy in fig. A.2 illustrates how a pitch can be heard even if its percieved frequency is absent just as one may see the word "PITCH" despite being presented with only a suggestive shadow.

Despite its psychoacoustic nature, it is nevertheless useful to objectively quantify pitch as a frequency. To do so, we first need some definitions. The difference between two frequencies

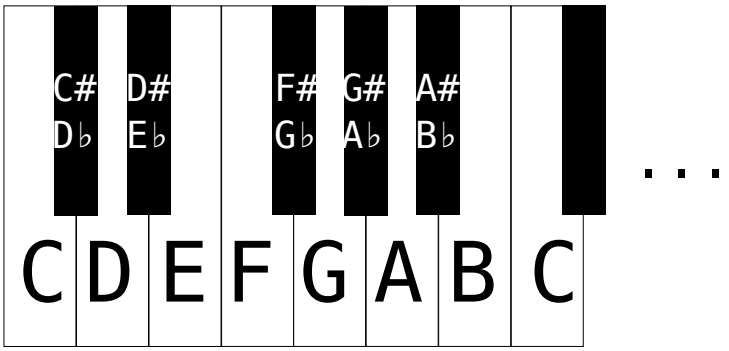


Fig. A.3 Illustration of an octave in the 12-note chromatic scale on a piano keyboard.

is called an **interval** and an **octave** is an interval corresponding to the distance between a frequency $f \in \mathbb{R}^+$ and its doubling $2f$ or halving $f/2$. Two frequencies spaced exactly an octave apart are perceived to be similar, suggesting that music is percieved on a logarithmic scale.

Most Western music is based on the **twelve-note chromatic scale**, which divides an **octave** into twelve distinct frequencies. The **tuning system** employed dictates the precise intervals between subdivisions, with **equal temperament tuning** (all subdivisions are equally spaced on a logarithmic scale) the most widely used in common practice music [37]. Under twelve-note equal temperament tuning, the distance between two consecutive subdivisions (1/12 of an octave) is called a **semitone** (British) or **half-step** (North American) and two semitones constitutes a **tone** or **whole-step**.

When discussing music, **note names** which enable succinct specification of a musical pitch are often employed. In **scientific pitch notation**, **pitch classes** which represent a pitch modulo the octave are specified by a letter ranging from A to G and optionally a single **accidental**. Pitch classes without accidentals are called **natural** and correspond to the white keys on a piano. Two accidentals are possible: sharps (#) raise the natural pitch class up one semitone and flats (♭) lower by one semitone. fig. A.3 illustrates how these pitch classes map to keys on a piano.

Since pitch classes represent equivalence class of frequencies spaced an integral number of octaves apart, unambiguously specifying a pitch requires not only a pitch class but also an octave. In scientific pitch notation, this is accomplished by appending an octave number to a pitch class letter (see fig. A.4). Together, a pitch class and octave number uniquely specify the notation for a pitch. On sheet music, the pitch of a note is indicated by its vertical position with respect to the **stave** (the five horizontal lines and four spaces).

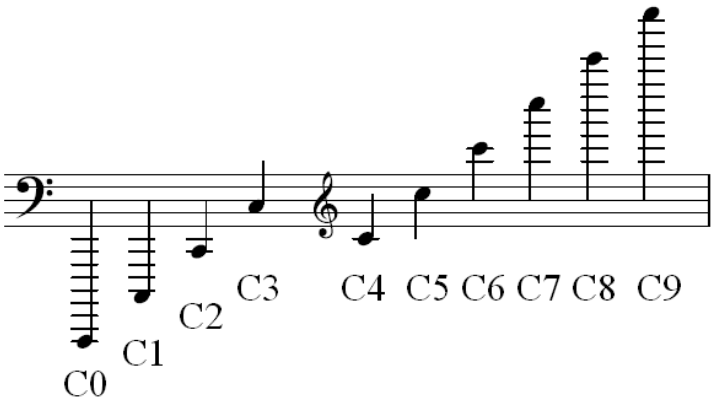


Fig. A.4 Scientific pitch notation and sheet music notation of *C* notes at ten different octaves.

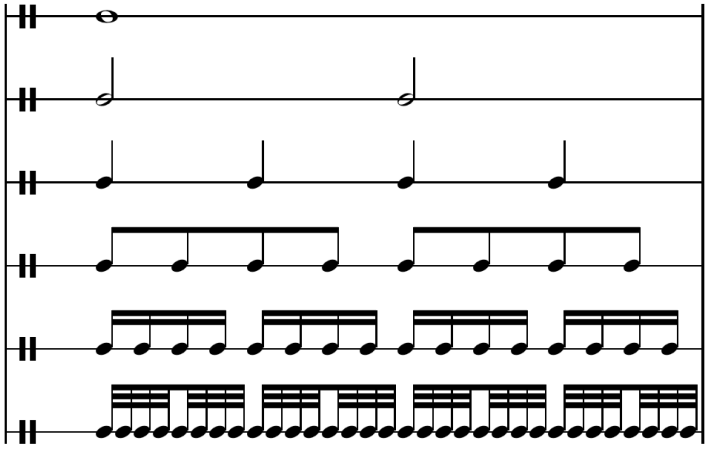


Fig. A.5 Comparison of various note durations [23]

A.1.2 Duration

In addition to pitch, a note also possesses a **duration**. The duration of a note indicates how long it is to be played and is measured in fractions of a **whole note** (American) or **semi-breve** (British). Perhaps the most common duration is a **quarter-note** (American) or **crotchet** (British). Other note durations are also possible and the most common along with their notation in sheet music are enumerated in fig. A.5. The relationship between durations and physical time intervals is given by the **tempo**, which is usually denoted near the start of the piece in beats per minute.

A.1.3 Offset, Measures, and Meter

The final property a note possess is an **offset** indicating the time at which a note is articulated. The offset is measured with respect to a fixed reference point in time, such as the start of a score.

Another common reference point for measuring offsets is with respect to the preceding **bar**. Bars are denoted by vertical lines through the stave in sheet music and are used to subdivide a piece into smaller temporal units called **measures**. Except for the notes preceeding the first bar (*i.e.* the **anacrusis**), most measures within a score all have the same duration.

In fig. A.1 on page 100, the crotchet preceding the first bar provides an example of an anacrusis. Notice that all other measures in the score are four crotchets in duration. In addition, observe that the offsets of notes within a measure is highly repetitive. There is always a note articulated on the first crotchet of a measure and articulations occuring between crotchets (*i.e.* quavers) are only present the last two crotchets of a measure.

This repetition of the same pattern of offsets across multiple measure helps establish a periodic pattern of strong and weak beats, a concept known as **meter** [81]. Meter is implied in Western music, where bars establish perodic measures of equal length [67]. The meter of a score provides information about musical structure which can be used to predict chord changes and repepetition boundaries [25].

A.1.4 Piano roll notation

Figure A.6 shows the the same score from fig. A.1 on page 100 in **piano roll notation**, a format which is convenient for visualization purposes. The horizontal and vertical axes represent time and pitch respectively. A solid horizontal bar signifies the presence of a note at the corresponding pitch and offset and the length of the bar corresponds to the note's duration.

A.2 Tonality in common practice music

Tonality refers to "the orientation of melodies and harmonies towards a referential (or **tonic**) pitch class" [76]. One way to characterize tonality is with **scales**, which defines a subset of pitch classes that are "in key" with respect to the tonic. Table A.1 shows the pitch intervals between adjacent pitch classes within two important scales: the **major** and the **minor**. The choice of tonic and scale is collectively referred to as the **key**.

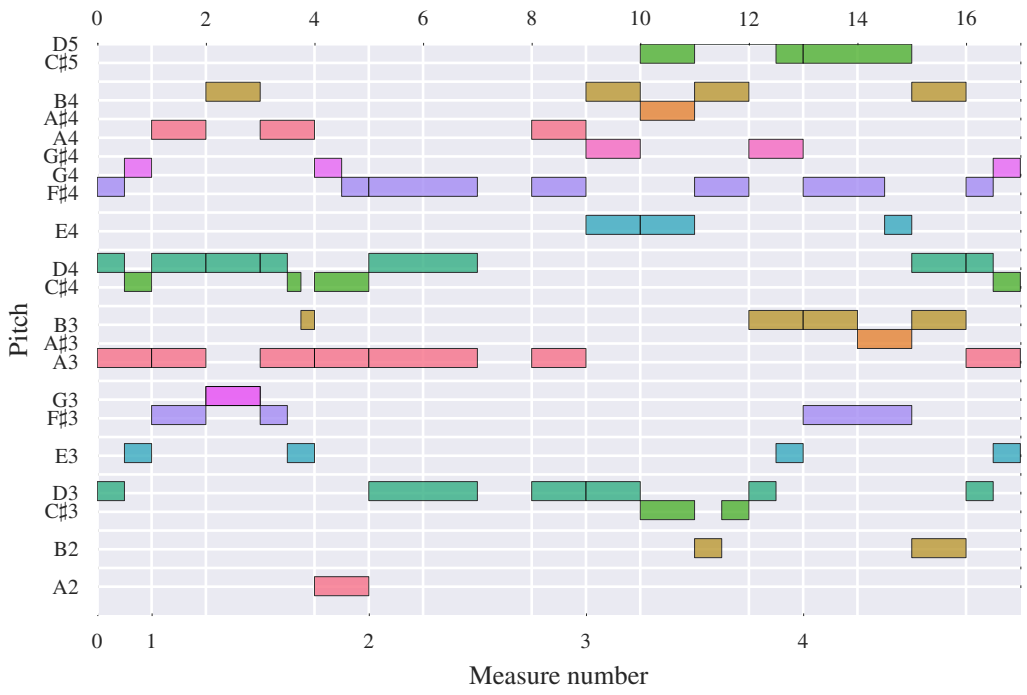


Fig. A.6 Piano roll notation of the music in fig. A.1

Table A.1 Pitch intervals for the two most important keys [53]. The pitches in a scale can be found by starting at the tonic and successively offsetting by the given pitch intervals.

Key	Pitch Intervals (semitones)
Major (Ionian, I)	+2, +2, +1, +2, +2, +2
Minor (Aeolian, VI)	+2, +1, +2, +2, +1, +2

A.2.1 Polyphony, chords, and chord progressions

Whereas **monophonic** music is characterized by the presence of a single **part** sounding at most one note at any given time, **polyphonic** music contains multiple parts potentially sounding multiple pitches at the same time. Just as notes form the basis of monophonic music, chords are the fundamental building blocks for polyphonic music.

A.2.2 Chords: basic units for representing simultaneously sounding notes

A **chord** is a collection of three or more pitches all sounding simultaneously [111]. In Western classical music, they typically consist of a **root note** whose pitch class forms a base from which successive notes are built upon. The intervals between the pitch classes in a chord are

Table A.2 Common chord qualities and their corresponding intervals [53]

Chord quality	Intervals from root pitch class
Major	+4, +7
Major 6th	+4, +7, +8
Major 7th	+4, +7, +11
Minor	+3, +7
Minor 6th	+3, +7, +9
Minor 7th	+3, +7, +10
Dominant 7th	+4, +7, +10
Augmented	+4, +8
Diminished	+3, +6
Diminished 7th	+3, +6, +9
Half-diminished 7th	+3, +6, +10

commonly labeled using **qualities**, which are invariant across octaves. Different realizations of the same chord (*e.g.* octave choices for each pitch class) are called **voicings**.

table A.2 lists some common chord qualities and their corresponding intervals from the root note. Chord names are given as a root pitch class followed by a quality, for example: *C* major, *A* minor, or *G* half-diminished 7th.

The lowest note in a chord is called the **bass** note and is oftentimes the root note. However, alternative voicings called **inversions** can place the root note on a different octave and cause the bass and root notes to differ.

A.2.3 Chord progressions, phrases, and cadences

Sequences of chords are called **chord progressions**, which are oftentimes grouped with adjacent progressions into coherent units called **phrases**. Many psychoacoustic phenomena such as stability, mood, and expectation can be attributed choice of chord progressions and phrase structure. For example, chord progressions can be used to create **modulations** which transition the music into a different key.

Analyzing chord progresssions involves a degree of subjectivity as chords can be overlapping and contain extraneous notes or involve uncommon chord qualities. A common method for analyzing chord progressions is **Roman numeral analysis**, where I is used for denoting the tonic pitch class, successive Roman numerals for successive pitch classes in the key, and capitalization is used to distinguish major and minor qualities. For example, the chord progression *C* major – *A* minor – *D* major 7th – *G* major in the *C* major key would be represented in Roman numerals as I – ii – IImaj7 – V.

A common use case for Roman numeral analysis is identifying and classifying chord progressions called **harmonic cadences**, which are commonly used for effects such as eliciting a sense of incompleteness [78] or establishing a sense of conclusion at the end of phrases [111]. The most important cadences include:

Perfect cadence : V – I. The perfect cadence is described by Randel [111] as "a microcosm of the tonal system, and is the most direct means of establishing a pitch as tonic. It is virtually obligatory as the final structural cadence of a tonal work"

Imperfect cadence : Any cadence ending on V, including I–V, ii–V, IV–V, V–V, and vi – V. The imperfect cadence sounds incomplete and is considered a **weak** cadence which call for continuation [78]

Interrupted cadepnce : V–vi. Also considered a weak cadence which invokes a "hanging" sensation prompting continuation [107].

A.2.4 Transposition invariance

Notice that the discussion thus far has remained ambiguous on the choice of tonic. This is intentional: most of the concepts discussed do not depend on the choice of tonic. When discussing tonality, the scale was defined using intervals relative to a choice of tonic. Similarly, Roman numeral analysis of chord progressions and cadences is also conducted relative to a tonic. Neither the scale nor the Roman numeral analysis is affected when a score is transposed by an arbitrary pitch interval.

The **transposition invariance** of chord progressions and keys is an important property of music. It enables us to offset an entire score by an arbitrary pitch interval without affecting many important psychoacoustic qualities.

B

Appendix B: An introduction to neural networks

This chapter provides background at a more elementary level than section 2.1 on page 27. Its goal is to sufficiently educate readers unfamiliar with recurrent neural networks such that the remainder of our work can be understood.

B.1 Neurons: the basic computation unit

Neurons are the basic abstraction which are combined together to form neural networks. A **neuron** is a parametric model of a function $f : \mathbb{R}^D \to \mathbb{R}$ from its D-dimensional input $\boldsymbol{x}$ to its output y. Our neurons will be defined as

$$f(\boldsymbol{x}) \coloneqq \sigma(\langle \boldsymbol{w}, \boldsymbol{x} \rangle) \tag{B.1}$$

which can be viewed as an inner product with **weights** $\boldsymbol{w}$ to produce an **activation** $z \coloneqq \langle \boldsymbol{w}, \boldsymbol{x} \rangle \in \mathbb{R}$ which is then squashed to a bounded domain by a non-linear **activation function** $\sigma : \mathbb{R} \to [L, U]$. This is visually depicted in fig. B.1, which also makes apparent the interpretation of weight w_i as the sensitivity of the output y to the input x_i.

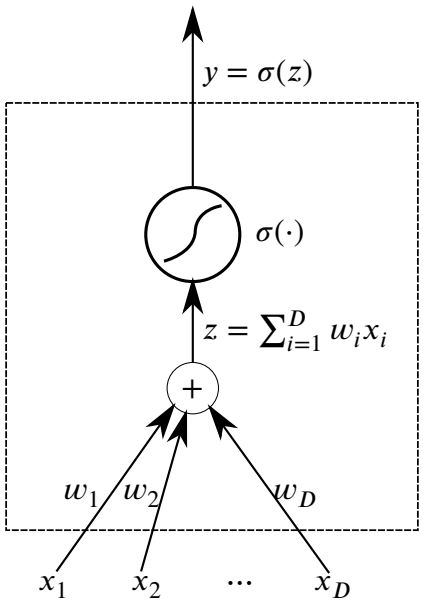


Fig. B.1 A single neuron first computes an activation z and then passes it through an activation function $\sigma(\cdot)$

B.2 Feedforward neural networks

Multiple neurons may share inputs and have their outputs concatenated together to form a **layer** modelling a multivariate functions $f : \mathbb{R}^{D_{\text{in}}} \to \mathbb{R}^{D_{\text{out}}}$. Multiple layers can then be composed together to form a **feedforwd neural network**.

Although a single hidden layer is theoretically sufficient for a universal function approximator [34], the number of hidden units to guarantee reported theoretical bounds are usually infeasibly large. Instead, recent work in **deep learning** has shown that deep models which contain many hidden layers can achieve strong performance across a variety of tasks [12].

The improved modeling capacity gained by composing multiple layers is due to the composition of multiple non-linear activation functions. In fact, it is easy to show that removing activation functions would make a deep network equivalent to a single matrix transform: let $\boldsymbol{W}_{l,l+1}$ denote the weights between layers l and $l+1$. The original neural network computes the function

$$\sigma\left(\boldsymbol{W}_{L,L-1}\sigma\left(\boldsymbol{W}_{L-1,L-2}\cdots\sigma\left(\boldsymbol{W}_{2,1}\boldsymbol{x}\right)\cdots\right)\right) \tag{B.2}$$

After removing the activation functions σ, we are left with

$$\boldsymbol{W}_{L,L-1}\boldsymbol{W}_{L-1,L-2}\cdots\boldsymbol{W}_{2,1}\boldsymbol{x} = \boldsymbol{x} = \tilde{\boldsymbol{W}}\boldsymbol{x} \tag{B.3}$$

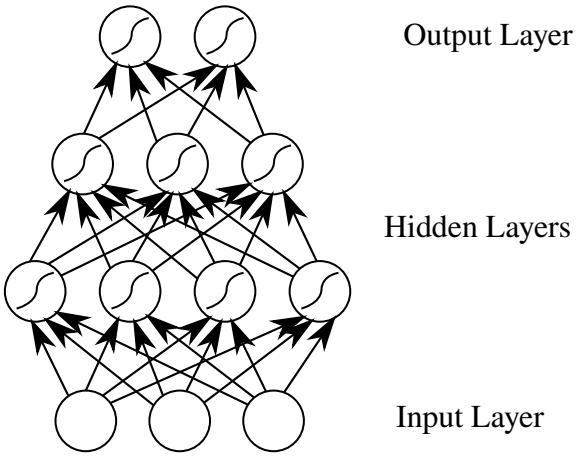


Fig. B.2 Graph depiction of a feedforward neural network with 2 hidden layers

where $\tilde{\boldsymbol{W}} = \left(\prod_{i=1}^{L-1} \boldsymbol{W}_{i,i+1}\right)$ is a matrix transform computing the same function as the neural network with activation functions removed.

B.3 Recurrent neural networks

While feedforward neural networks provide a flexible model for approximating arbitrary functions, they require a fixed-dimension input $\boldsymbol{x}$ and hence cannot be directly applied to sequential data $\boldsymbol{x} = (\boldsymbol{x}_t)_{t=1}^{T}$ where T may vary.

A naive method for extending feedforward networks would be to independently apply a feedforward network to compute $\boldsymbol{y}_t = f(\boldsymbol{x}_t\boldsymbol{\theta})$ at each timestep $1 \leq t \leq T$. However, this approach is only correct when each output $\boldsymbol{y}_t$ depends only on the input at the current time $\boldsymbol{x}_t$ and is independent of all prior inputs $\{\boldsymbol{x}_k\}_{k<t}$. This assumption is false in musical data: the current musical note usually is highly dependent on the sequence of notes leading up to it.

This shortcoming motivates **recurrent neural networks** (RNNs), which generalize feedforward networks by introducing time-delayed recurrent connections between hidden layers (Elman networks [46]) or from the output layers to the hidden layers (Jordan networks [79]). Mathematically, an (Elman-type) RNN is a discrete time dynamical system commonly parameterized as:

$$\left.\begin{aligned} \boldsymbol{h}_t &= \boldsymbol{W}_{xh}\sigma_{xh}\left(\boldsymbol{x}_t\right) + \boldsymbol{W}_{hh}\sigma_{hh}\left(\boldsymbol{h}_{t-1}\right) \\ \boldsymbol{y}_t &= \boldsymbol{W}_{hy}\sigma_{hy}\left(\boldsymbol{h}_t\right) \end{aligned}\right\} \quad \text{RNN Dynamics} \tag{B.4}$$

(B.5)

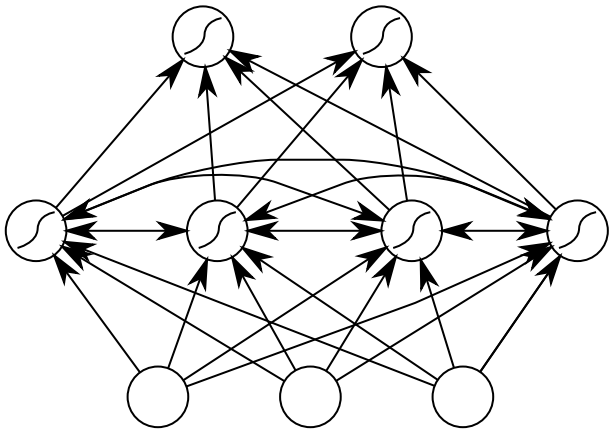


Fig. B.3 Graph representation of an Elman-type RNN.

where $\sigma_{..}(\cdot)$ are activation functions acting element-wise and $\theta = \{\boldsymbol{W}_{xh}, \boldsymbol{W}_{hh}, \boldsymbol{W}_{hy}\}$ are the learned parameters. fig. B.3 provides a graphical illustration of such a network. Notice that apart from the edges between hidden nodes, the network is identical to a regular feedforward network (fig. B.2).

To apply the RNN over an input sequence $\boldsymbol{x}$, the activations of the hidden states are first initialized to an initial value $\boldsymbol{h} \in \mathbb{R}^{D_h}$. Next, for each timestep t the hidden layer activations are computed using the current input $\boldsymbol{x}_t$ and the previous hidden state activations $\boldsymbol{h}_{t-1}$. This motivates an alternative perspective on RNNs as a template consisting of a feedforward network with inputs $\{\boldsymbol{x}_t, \boldsymbol{h}_{t-1}\}$ (see fig. 2.1 on page 29) replicated across time t.

9 Graveyard

Every aspiring music theorist is at some point tasked with composing simple pieces of music in order demonstrate understanding of the harmonic rules of Western classical music. These pedagogical exercises often include harmonization of chorale melodies, a task which is viewed as sufficiently constrained to allow a composer's basic technique to be judged. A generative model for music scores can be applied to this task by conditioning on the melody line and sampling the conditional distribution for possible harmonizations.

A more difficult task is automatic composition, where the composer is tasked with producing an original composition of a particular musical style. The open nature of this task enables a composer to demonstrate both their understanding of music theory as well as their creativity. However, this lack of constraints and loose definition of musical style makes it more difficult to evaluate the quality of the output. To apply a generative model towards this task, we can train the model to assign larger probability mass to stylistically similar scores and then sample the model to generate a novel composition.

9.1 Neural Networks

A common choice is the logistic function $\sigma(z) = \frac{1}{1+\exp -z}$, which squashes $\boldsymbol{y} \in [0, 1]$. Other choices include $\sigma = \tanh$, in which case $[L, U] = [-1, 1]$.

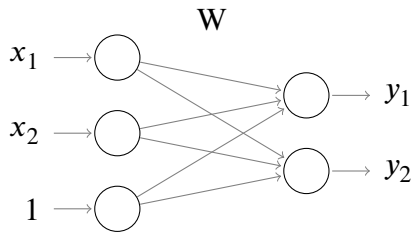


Fig. 9.1 Single feedfoward neural network layer

It is common to represent feedforward neural networks as directed acyclic graphs (

fliang: CITE: fig:nn-layer

). Here, each node denotes a data value and an edge from s to t notates that the value at s is used to compute the value at t.

Multiple layers can be composed together by treating the outputs from the previous layer as the inputs to the next layer.

fliang: CITE: fig:ffw-nn

illustrates this on a 2-layer feedforward neural network where the outputs of the first layer are used as the inputs to the second layer (*i.e.* $x^{(1)} = y^{(0)}$).

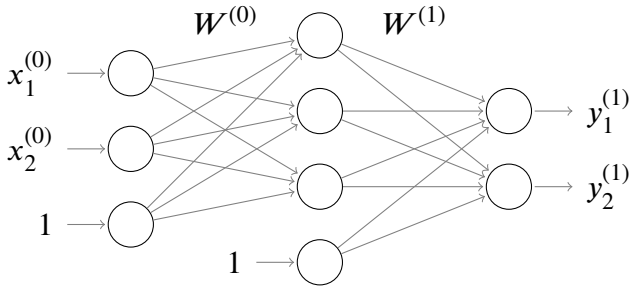


Fig. 9.2 2-layer feedforward neural network

When discussing neural networks with $L \geq 1$ layers, we will use $\boldsymbol{x}^{(i)}$, $\boldsymbol{W}^{(i)}$, $\boldsymbol{z}^{(i)}$, and $\boldsymbol{y}^{(i)}$ to refer to the inputs, weights, activations, and outputs of the ith layer. The activation function σ is understood to act elementwise when applied to a vector. For adjacent layers $i, i+1$, we have $\boldsymbol{x}^{(i+1)} = \boldsymbol{y}^{(i)}$. $\boldsymbol{x}^{(0)}$ and $\boldsymbol{y}^{(L)}$ are the inputs and outputs respectively of the entire network.

The non-linearity introduced by the activation function σ is paramount for enabling neural networks to model a broad variety of functions.

fliang: If activation functions are removed, then a neural network can only model affine transformations.

Modeling probability distributions

A neural network can be used to model the distribution of a categorical random variable o by treating the final layer activations $\boldsymbol{z}^{(L)}$ as the energies of a Boltzmann distribution (*i.e.* softmax). This implies a probability mass function on o given by

fliang: CITE: eq:softmax

.

$$P(o = k|\boldsymbol{z}^{(L)}) = \frac{\exp -z_k^{(L)}}{\sum_j \exp -z_j^{(L)}} \tag{9.1}$$

Efficient graident computations through back-propogation

Feed-forward neural networks are trained using back-propogation, an efficient algorithm which consists of a forward pass to compute activations followed by back-propogation of partial derivatives expanded according to the chain rule

fliang: cite backprop

. At the heart of back-propogation is the **computation graph** of a a model: a directed acyclic graph where each node represents a differentiable function that can compute its outputs and Jacobian given inputs and activations

fliang: cite theano

. By representing only the dependencies between intermediate values, the sparsity imposed by the computation graph enable back-propogation to ignore irrelevant cross-derivatives and efficiently compute global gradients from local computations.

Training of recursive neural networks is typically performed using backpropogation through time (BPTT)

fliang: Cite

, a technique computationally equivalent to feedforward training of the unrolled computation graph. This is easily seen: unrolling of a RNN yields a feed-forward structure where the standard back-propogation algorithm applies.

Vanishing gradients

The solution is to rewrite

fliang: CITE: eq:ht-from-ht-1

such that

fliang: CITE: eq:prod-hi

does not vanish/explode for large $t - k$. One possibility would be

$$h_t = h_{t-1} + \theta_x x_t \tag{9.2}$$

However, this solution is unsatisfactory as all hidden state dynamics have been removed.

Training with back-propogation

Training neural networks is achieved using gradient descent methods, which optimize parameters $\theta = \{W^{(i)} : 1 \leq i \leq L\}$ to minimize some loss function $L(\boldsymbol{z}^{(L)}_{1:N}, \hat{o}_{1:N})$ between the network outputs $\boldsymbol{z}^{(L)}_{1:N}$ and the true labels $\hat{o}_{1:N}$. For probabilistic classification, a common choice is to assume independence across training examples and use **cross-entropy loss** (

fliang: CITE: eq:cross-entropy-loss

):

$$\begin{aligned} L(\boldsymbol{z}^{(L)}_{1:N}, \hat{o}_{1:N}) &= \sum_{i=1}^{N} L(\boldsymbol{z}^{(L)}_i, \hat{o}_i) && \text{Independence across samples} \\ &= \sum_i \sum_k \delta_{\hat{o}_i,k} \log \frac{1}{P(o = k | \boldsymbol{y}^{(L)}_i)} \end{aligned} \tag{9.3}$$

Gradient descent proceeds by using the Jacobian (*i.e.* gradient) $\nabla_\theta L(\boldsymbol{z}^{(L)}_{1:N}, \hat{o}_{1:N})$ to iteratively update the network parameters using successive first-order approximations (

fliang: CITE: eq:nn-training-iteration-scheme

).

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \left[\nabla_\theta L(\boldsymbol{z}^{(L)}_{1:N}, \hat{o}_{1:N}) \right]_{\theta=\theta^{(t)}} \tag{9.4}$$

Variants of

fliang: CITE: eq:nn-training-iteration-scheme

which adaptively set the step size η_t or incorporate/estimate the Hessian $\nabla^2_\theta L(\cdot, \cdot)$ can yield performance when applied to neural network training. However, their discussion is out of scope.

fliang: Discuss RMSprop?

To apply

fliang: CITE: eq:nn-training-iteration-scheme

, the gradient $\nabla_\theta L(\boldsymbol{z}^{(L)}_{1:N}, \hat{o}_{1:N})$ must be computed. This can be accomplished using **back-propogation**

fliang: cite

, an algorithm which exploits the independence structure to avoid unnecessary computations and make gradient computations tractable.

Let $\delta^{(l)}_j = \frac{\partial L(\boldsymbol{z}^{(L)}_{1:N}, \hat{o}_{1:N})}{\partial z^{(l)}_j}$ be the partial derivative of the loss with respect to the jth activation of layer l. For the final Lth layer, cross-entropy loss with a Boltzmann distribution yields

fliang: CITE: eq:cross-entropy-loss

$$\begin{aligned}
\delta^{(L)}_j &= -\sum_{i=1}^{N}\sum_{k}\frac{\partial}{\partial z^{(L)}_j}\delta_{\hat{o}_i,k}\log P(o=k|\boldsymbol{z}^{(L)}_i) && \text{CITE HERE} \\
&= \sum_{i=1}^{N}\left(P(o=k|\boldsymbol{z}^{(L)}_i) - y_i\right) && \text{Softmax derivative}
\end{aligned}$$

For earlier layers $l < L$, we have

$$\begin{aligned}
\delta^{(l)}_j &= \sum_k \frac{\partial L(\boldsymbol{z}^{(L)}_{1:N}, \hat{o}_{1:N})}{\partial z^{(l+1)}_k}\frac{\partial z^{(l+1)}_k}{\partial z^{(l)}_j} && (9.5) \\
&= \sum_k \delta^{(l+1)}_k \frac{\partial}{\partial z^{(l)}_j}\left(\boldsymbol{W}^{(l+1)}[\sigma(z^{(l)}), 1]^\intercal\right)_k && (9.6) \\
&= \sum_k \delta^{(l+1)}_k \boldsymbol{W}^{(l+1)}_{k,j}\sigma'(z^{(l)}_j) && (9.7)
\end{aligned}$$

This expression can be vectorized using the Hadamard product (elementwise multiplication), which improves performance due to CPU cache locality and coalesced memory loads:

fliang: DO THIS

$$\circ \qquad (9.8)$$

This recursion can be iterated until $l \to 0$.

The back-propogation algorithm consists of two steps:

1. **Forward pass**: Using current model parameters $\theta^{(t)}$, feed the data into the network to compute the activations $\boldsymbol{z}^{(l)}$, $1 \leq l \leq L$

2. **Backward pass**: Recursively iterate

fliang: CITE: eq:backprop

to compute $\boldsymbol{\delta}^{(l)}$, $1 \leq l \leq L$ using the activations $\boldsymbol{z}^{(l)}$ obtained from the forward pass

After the backwards pass, gradients with respect to model parameters are easily obtained

$$\frac{\partial L}{\partial W_{i,j}^{(l)}} = \sum_k \frac{\partial L}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial W_{i,j}^{(l)}} \tag{9.9}$$

$$= \sum_k \delta_k^{(l+1)} z_j^{(l)} \tag{9.10}$$

Some appealing properties of backpropogation:

- Efficient exploitation of the computation graph: chain rule expansions are constrained by the computation graph, improving efficiency because factors which don't contribute to a given $\delta^{(l)}$ are neglected in the recursion

- Implementation using local rules: the forward/backward pass at any layer l only requires knowledge of $z^{(l)}$, $\delta^{(l+1)}$, and the derivative of the activation σ'. As all these quantities are localized to one layer, this permits modular implementations where a node which can be back-propped through needs only implement a `forward()` method which computes activations given inputs and a `backward()` method which computes $\delta^{(l)}$ given activations.

fliang: Talk about how localization gives rise to computation graph and autodiff

9.2 RNNs

The advantages of RNNs over feedforward networks include:

- Ability to handle variable-length inputs: the RNN can be unrolled an arbitrary number of times to accomodate inputs $\boldsymbol{x}$ of different length

- Fixed dimension embeddings: after processing the entirety of an input sequence, the state of the RNN can be used as a fixed dimension embedding representing the input

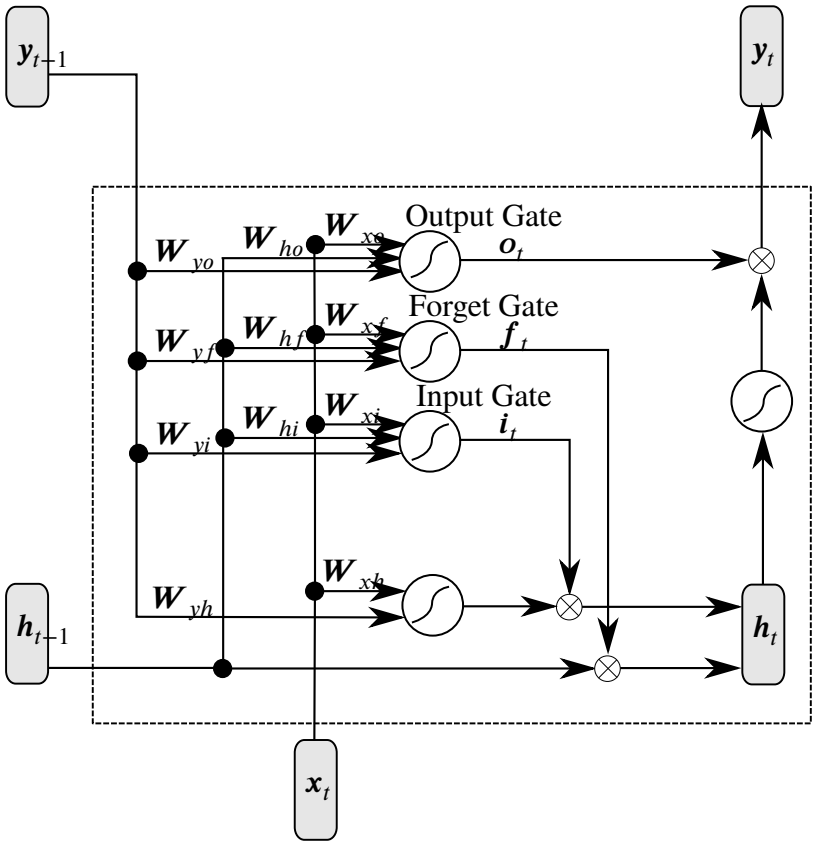


Fig. 9.3 Single LSTM unit

- Sequential processing: the order of $\boldsymbol{x}_{1:T}$ will affect the state trajectory $s_{1:T}$, enabling the model to capture time-dependent dynamics within the input sequences
- Memory over time: the state $s \in \mathbb{R}^D$ can take on an uncountably infinite number of values, allowing it to potentially act as memory which summarizes **all** of the input up to the current time

Comparison against HMMs

Hidden Markov Models (HMMs) are another popular probabilistic model for sequental data.

fliang: Define HMMs

While RNNs are similar HMMs in that both model the conditional distribution of next frames given the previous context. However, RNNs additionally pass along "hidden state" which summarizes contextual information from a potentially infinite context window.

9.3 Sequence probability modelling

Generating a "Bach-like" piece of music can be understood as drawing a random sample from a distribution over musical scores which is statistically similar to Bach's own compositions. Thus, we interpret the problem as one of **categorical sequence modeling**.

This type of problem has been well studied. In speech recognition, language models parameterizing distributions over sentences are used as priors to refine transcriptions.

However, since our model has to be able to generate Bach, we must be able to sample from it. This rules out a broad class of sequence models, including back-off N-grams and other interpolated language models.

Fortunately, low order N-grams and standard HMM-based models are sampleable and thus can be used as baselines.

9.4 Related work

[33] used `music21` to generate rich feature representations for music for downstream machine learning tasks.

The application of machine learning to music has a rich history. [69] describe a system to classify music into homogeneous styles. However, they focus on the discriminative task and do not consider how to generate novel scores.

In **Bach in a Box** [119], harmonic rules are collected in a database and then used to build rule-based neural networks. This enables encoding of prior knowledge as rules in the rulebase.

9.5 LSTM: background and motivation

Two prominent methods for training RNNs include real-time recurrent learning (RTRL) [113] and backpropogation through time (BPTT) [140]. [138] introduces truncated BPTT to address computational complexity when learning over very long sequences. Temporal difference [127] has also been proposed as a method for learning RNNs [50].

The first LSTM models, which did not include forget gates, was introduced in [72]. [56] later revised the LSTM model to include forget gates in order to prevent hidden states from growing indefinitely.

LSTM have been demonstrated to outperform traditional RNNs on a variety of tasks. [54] demonstrates a LSTM correctly recognizing 1000 instances from the context-free grammar $A^n B^n$ while an Elman RNN achieves only 20% accuracy.

Online adaptation at test time using a Kalman filter was described in [58]. [95] [93] refers to this as "dynamic evaluation."

[43] attempts to model meter by introducing time-delayed connections in [44]

9.5.1 Representation of music data

[96] discusses the importance of music representation, settling on **psychologically-based representations** of pitch, duration, and harmonic structure [118].

Many attempts to represent musical data have been investigated. Attempts which explicitly model harmonic structure include a Circle of Thirds representation [51] or overlapping subharmonics representation[85], both of which have been studied in the context of generative RNN models [51] [96]. Other representations attempt to model notions such as musical distance in terms of voice leading, orbifolds, and tuning lattices[134].

[49] introduce a LSTM model for jazz melodies which use separate units for notes and their durations.

The success of these methods are varied and it remains ambiguous if any is better. Furthermore,

9.6 Evaluation of models

[106] addresses difficulty in quantitative evaluation, suggesting the use of a learned critic in a manner similar to GANs [61]. In a later report, [105] attribute difficulty in evaluation due to lack aim: algorithmic composition, design of compositional tools, and computational modelling of musical styles or music cognition all have different motivations and should thus be evaluated differently.

[5] criticizes a musical Turing test as providing little data about how to improve the system, suggesting that listener studies using music experts may be more insightful.

9.7 Token-level embeddings

fliang: EXPERIMENT: redo these

Filter to notes

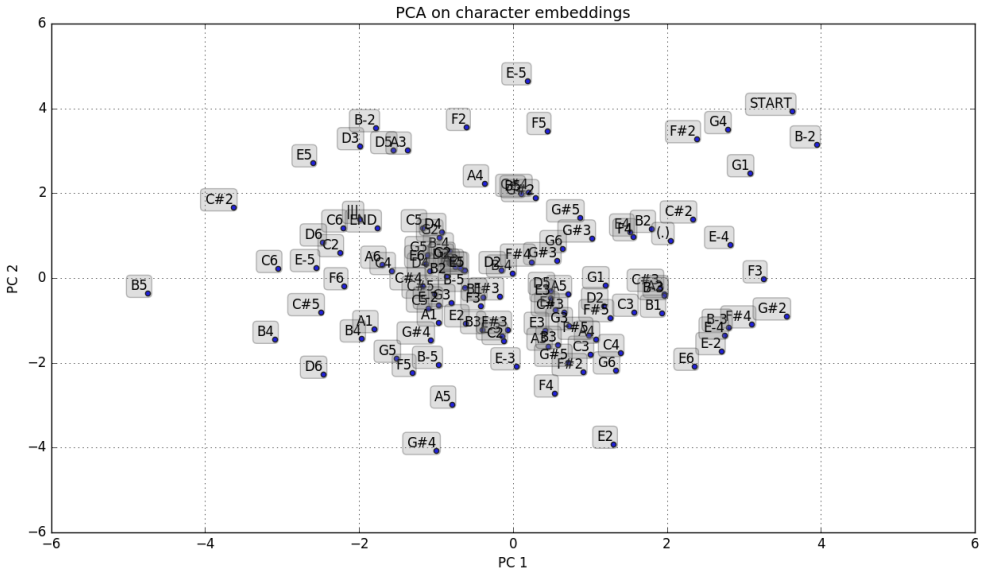


Fig. 9.4 PCA embedding of note tokens

9.7.1 Variable-length embeddings

fliang: EXPERIMENT: LSTM hidden state after consuming chord (chord boundary, do they cluster?), phrase (up to fermata, do similar phrases embed similarly), whole pieces (difficult to evaluate)

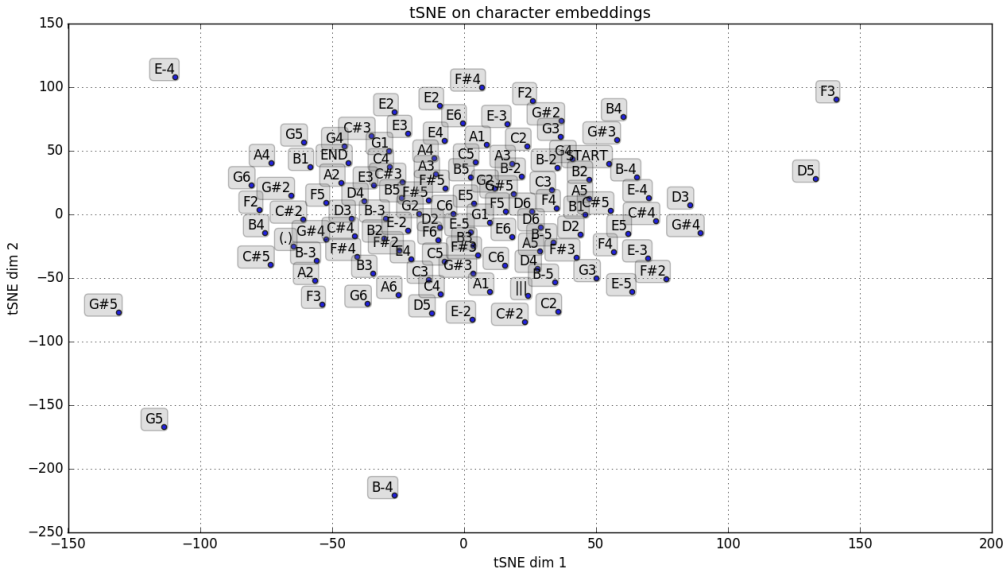


Fig. 9.5 tSNE embedding of note tokens