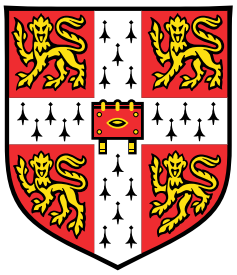
BachBot: Automatic stylistic composition of Bach chorales

Developing, analyzing, and evaluating a deep LSTM model of musical style



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, Luping and Yueli, who have supported me at all steps of my journey through life and academia. And to my sister, Dawn, whom I am much too sleep-deprived to write a dedication for but you know I love you regardless.

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: 11356

Signed:

Date:

Feynman Liang
August 2016

Acknowledgements

I would like to acknowledge Mark Gotham, my primary point of contact for anything music related. The time he spent teaching me music theory, helping me design experiments, and providing feedback on intermediate results was invaluable and greatly appreciated.

I would also like to acknowledge my industry sponsors, Matthew Johnson and Jamie Shotton from Microsoft Research Cambridge, for their role proposing the idea of BachBat, providing computing resources, and and giving me regular feedback on the progress of the project.

In addition, I would like to acknowledge my academic supervisor Bill Byrne for hissupport guiding me through the thesis writing proces.

Finally, I would like to acknowledge my proofreaders and reviewers, which include all of the aforementioned as well as Niole Nelson, Kyle Kastner, and Tom Nicholson.

Abstract

This thesis investigates Bach's music composition style using deep sequence learning. We develop *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, we **consciously avoid allowing prior assumptions impact model design**, opting instead to build systems that learn rather than ones which encode prior hypotheses. 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 which have learned to specialize to detect music-theoretic concepts such as chords and cadences without any prior knowledge or supervision. 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 **more than three times larger than the next-closest comparable study** [91]. We demonstrate that a voluntary participation study promoted over social media can generate responses from a significant number (165 at time of writing) of highly-skilled domain experts, and is infinitely more cost efficient. After evaluating BachBot on 721 participants, we found that participants could only differentiate BachBot's generated chorales from Bach's originals works only 9% better than random guessing.

In other words, **generating stylistically successful Bach chorales is more closed (as a result of BachBot) than open a problem**.

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
P^*	true probability distribution
$\tilde{P}$	distribution predicted by model
T	total number of timesteps in a sequence
$\boldsymbol{W}$	weight matrix
$\hat{\boldsymbol{x}}$	values of fixed tokens in a harmonization
$\boldsymbol{x}^*$	the optimal harmonization
$\tilde{\boldsymbol{x}}$	the proposed harmonization

Greek Symbols

α	multi-index of fixed tokens in harmonization
δ	Kroneker delta
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

$\boldsymbol{o}_t$ 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

OOV out of vocabulary

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 [60, 65, 39, 95] 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 21 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 analogous to that of language modelling in speech research. Unfortunately, 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 chapter 5 on page 35, 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 43 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 47 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 can yield high quality data. This finding is especially significant to other fields requiring human evaluation, such as machine translation, as it represents an alternative to the increasingly controversial Amazon MTurk[34] for human evaluation.

Finally, we summarize the conclusions from our work and suggest future directions for extension in chapter 8 on page 57.

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 69 for a quick primer and Piston [90] and Denny [30] 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 77 and consult Bengio [9] for further reference.

2.1 Recurrent neural networks

Our use of the term **recurrent neural network** (RNN) refers in particular to linear Elman-type RNNs [40] whose dynamics are described by eq. (2.1) on the following page (review chapter B if this is unfamiliar).

2.1.1 Notation

We begin by clarifying common notation and conventions used to describe 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 elementwise **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 10, $\boldsymbol{W}_{xf}$ 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 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 [40, 67, 61, 18, 71, 79], most share the same underlying structure and differ only in their implementation details of eq. (2.1). Encapsulating these differences within an abstraction 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 [40] with the memory cell indicated by a dashed box isolating the recurrent hidden state. The edges entering the memory cell ($\boldsymbol{x}_t$, $\boldsymbol{h}_{t-1}$) are the **memory cell inputs**

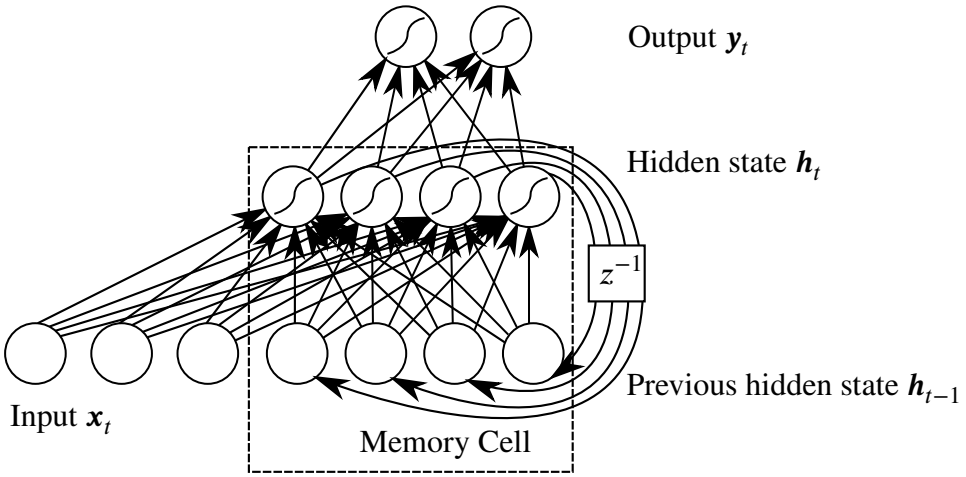


Fig. 2.1 An Elman-type RNN with a single hidden layer. The recurrent hidden state is illustrated as unit-delayed (denoted by z^{-1}) feedback edges from the hidden states to the input layer. The memory cell encapsulating the hidden state is also shown.

and the outgoing edges ($\boldsymbol{y}_t$, $\boldsymbol{h}_t$) are the **memory cell outputs**. In essence, the memory cell abstracts away differences across RNN variants in their implementation of eq. (2.1).

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 [39, 95] 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. [87], affirming the importance of stacking multiple layers in RNNs. The improved modelling 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 [9]. 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 illus-

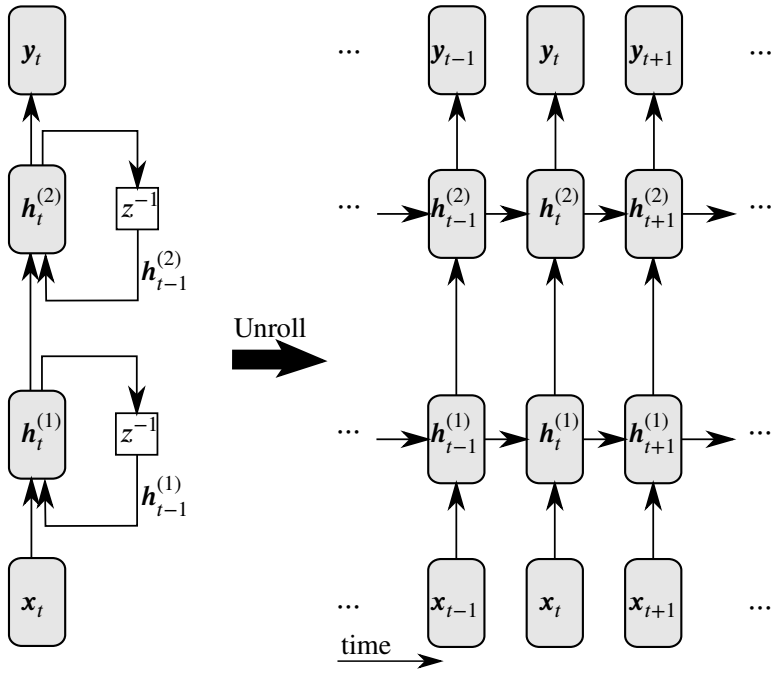


Fig. 2.2 Block diagram representation of a -layer RNN (left) and its corresponding DAG (right) after unrolling. The blocks labelled with $\boldsymbol{h}_t$ represent memory cells whose parameters are shared across all times t.

trated for a stacked 2-layer RNN in fig. 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 memory mechanism. Under this interpretation, updating the hidden state $\boldsymbol{h}_t$ can be viewed as **writing** information from the current inputs $\boldsymbol{x}_t$ to memory and producing the outputs $\boldsymbol{y}_t$ 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 [16, 74, 93]. While time-delayed recurrent hidden state connections appear to complicate matters initially, unrolling the RNN removes the time-delayed recurrent edges

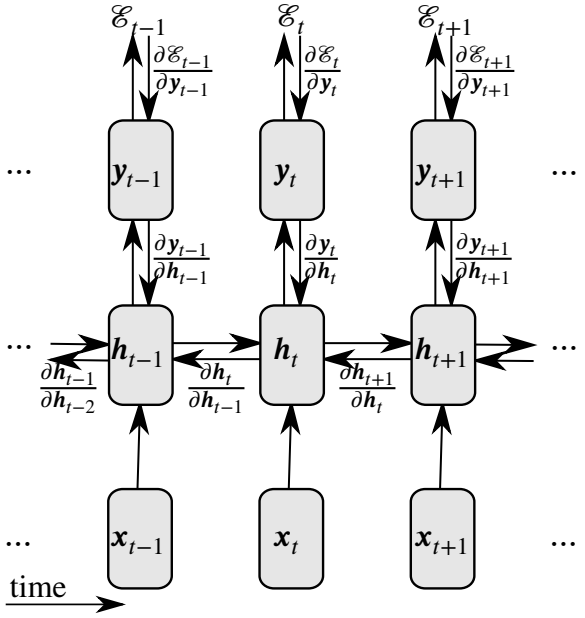


Fig. 2.3 The gradients accumulated along network edges in BPTT.

and converts the RNN into a DAG (*e.g.* fig. 2.2 on page 8) 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) [50], a method for training RNNs which applies backpropagation to the unrolled DAG.

Figure 2.3 shows how BPTT, just like regular backpropagation, 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 is hence cheaper to compute. However, whereas the depth of feedforward networks is fixed, the unrolled RNN's depth is equal to the input sequence length T and may introduce problems when T is very large.

Vanishing/exploding gradients

It is well known that naive implementations of memory cells often suffer from two problems also affecting very deep feedforward networks: the **vanishing gradient** and **exploding gradient** [11].

To illustrate the problem, express the computation represented by fig. 2.3 mathematically by applying the chain rule to the RNN dynamics equation (eq. (2.1) on page 6):

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

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

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

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 [86].

Both vanishing and exploding gradients are due to the product in eq. (2.4) exponentially growing or shrinking over long time-spans (*i.e.* $t \gg k$), preventing error signals to be transferred across long time-spans and learning of long-term dependencies. In section C.1 on page 81 we prove that a sufficient condition for vanishing gradients is:

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

where $\| \cdot \|$ is the matrix operator norm (see eq. (C.1) on page 81), $\boldsymbol{W}_{hh}$ is as defined in eq. (2.1) on page 6, 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 [22]:

> 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 [61], LSTM solves the vanishing

gradient problem by enforcing **constant error flow** on eq. (2.4), that is

$$\forall t, \forall \boldsymbol{h}_t \,:\, \boldsymbol{W}_{hh}^{\mathsf{T}} \sigma'_{hh}(\boldsymbol{h}_t) = \boldsymbol{I} \tag{2.6}$$

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

As a consequence of constant error flow, eq. (2.4) on page 10 becomes

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

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) with respect to $\boldsymbol{h}_t$ 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:

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

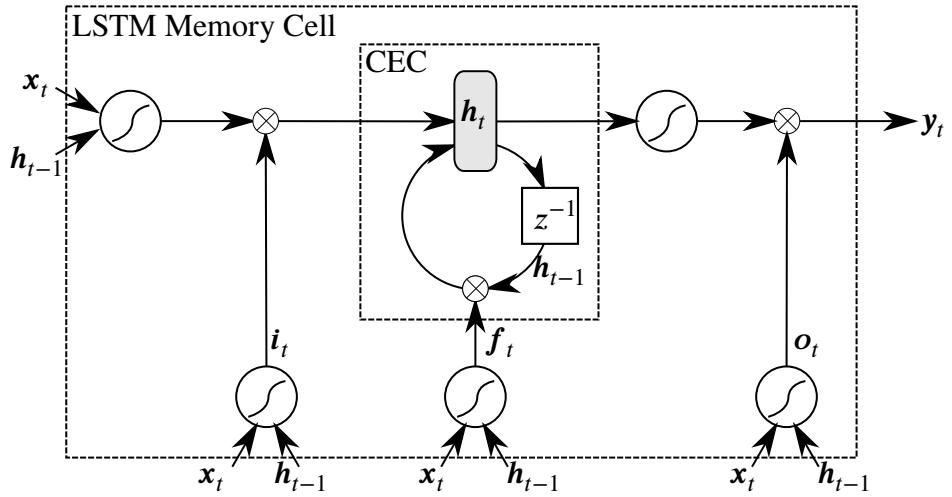


Fig. 2.4 Schematic for a single LSTM memory cell. Notice how the gates $\boldsymbol{i}_t$, $\boldsymbol{o}_t$, and $\boldsymbol{f}_t$ control access to the constant error carousel (CEC).

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 time dependent. This permits interpreting the gates as a mechanism enabling LSTM to learn which error signals to trap in the CEC and when to release them [61], allowing 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 fig. 2.4 as LSTM with "peephole connections" [47]. We will use LSTM to refer to the model as described above.

Practicalities for successful applications of LSTM

Many successful applications of LSTM [32, 113, 87] employ some common practical techniques. Perhaps most important is **gradient norm clipping** [78, 86] where the gradient is scaled or clipped whenever it exceeds a threshold. This is necessary because while 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** [60] is commonly applied between stacked memory cell layers to regularize the learned features and prevent co-adaptation [114]. Additionally, **batch normalization** [65] of memory cell hidden states is also commonly done 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 [101]. 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 back-propagating error signals a fixed number of timesteps back in the unrolled network, a technique known as **truncated BPTT** [111]. As the hidden states in the unrolled network have already been exposed to many previous timesteps, learning of long range structure is still possible with truncated BPTT.

3
Related Work

3.1 Prior work in automatic composition

In a review by Toiviainen [106], automatic composition methods are broadly classified as either symbolic (*e.g.* rule-based expert systems) or connectionist (*e.g.* neural networks). While our research falls strongly within the connectionist category, we provide review methods from both categories.

3.1.1 Symbolic rule-based methods

Symbolic methods have been prevalent since the 1960s [104] and are appealing because of their high degree of interpretability. As described by Todd [105], 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."

At the heart of many rule-based systems is a collection of rules which are (recursively) applied to ultimately yield musical notes. While the earliest rule-based systems required manual specification of rules [35, 27], later works utilized techniques such as association

rule mining [97], grammatical inference [27, 91], or constraint logic programming [107] to automatically derive new rules or learn them from data.

Experiments in Music Intelligence (EMI) by Cope [24, 23] is one of the first rule-based composition systems which achieved automatic stylistic composition. Using a hand-crafted grammar and an augmented transition network parser [109], the system was capable of producing music to a particular genre or author, suggesting that the rules extracted by the system can capture a sense of musical style. The more recent *Emmy* and *Emily Howell* projects [25, 26] extend EMI by using it as a database of compositions to recombine and build novel compositions from.

While symbolic methods permit straightforward incorporation of domain-specific knowledge and a offer high degree of interpretability, they are inherently biased by their creators' subjective theories on harmony and music cognition. Furthermore, specification of hand-crafted rules requires music expertise and the rules may not generalize across different tasks. Additionally, rule-based methods are brittle to even small amounts of distortion and noise, making them unsuitable for noisy applications. Furthermore, symbolic methods limit creativity by disallowing any form of deviation from the defined rules.

3.1.2 Early connectionist methods

Connectionism, also known as parallel distributed processing, refers to systems built from several simple processing units connected in a network and acting in cooperation [55]. Unlike rule based systems, the connectionist paradigm replaces strict rule-following behaviour with regularity-learning and generalization [33].

The earliest connectionist music models utilized note-level Jordan RNNs [67] for melody generation and harmonization tasks [104, 105, 12]. While they achieved "varying degrees of success" [54], their creators did not conduct any rigorous evaluations.

The next generation of models utilized prior knowledge of music theory to inform their designs. Mozer's CONCERT [81] system is a BPTT-trained RNN which models music at two levels of resolution (notes and chords) and utilizes domain-specific representations for notes [96] and chords [73]. Similarly, HARMONET [58] also applies domain-specific knowledge to break down the prediction pipeline into first predicting the Roman numeral skeleton of a piece followed by chord expansion and ornamentation. MELONET [41, 63] builds on top of HARMONET an additional motif classification sub-network.

A major criticism of these early models is their highly specialized domain-specific architectures. Despite the connectionist philosophy of learning from data rather than imposing prior constraints, the models developed are highly influenced by prior assumptions about the structure of music and incorporate significant amounts of domain-specific knowledge.

Additionally, these models had difficulties learning the long-term dependencies required for plausible phrasing structure and motifs. Mozer describes CONCERT as being able to reproduce 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 [81]). This problem of learning long-term dependencies can likely be attributed to the memory cells used by earlier models, which did not protect against vanishing gradients.

3.1.3 Modern connectionist models

The invention of LSTM in 1997 by Hochreiter and Schmidhuber [61] brought on a new generation of connectionist models which utilized more sophisticated memory cell implementations. Experiments demonstrated that LSTM possessed many properties desirable for music applications, such as superior performance learning grammatical structure [46], capability to measure time intervals between events [47], and ability to learn to produce self-sustaining oscillations at a regular frequency Gers, Schraudolph, and Schmidhuber [48]. Franklin [44] evaluated multiple memory cells 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 [38] and Eck and Schmidhuber [36], which used an LSTM to model blues chord progressions and another LSTM to model melody lines given chords. The authors reported that LSTM can learn long term music structure such as repeated motifs without explicit modelling, an improvement over earlier systems such as HARMONET by Feulner and Hörnel [41] where motifs were explicitly modelled. However, Eck and Schmidhuber [38] used a severely constrained music representation which quantized to eight notes, neglected the octave numbers for pitch classes, limited the model to 12 possible chords, and had "no explicit way to determine when a note ends" Eck and Schmidhuber [38].

The current state of the art in polyphonic modelling is split between the RNN-RBM [13] and RNN-DBN [49] depending on the dataset used for evaluation. However, both models requires an expensive contrastive divergence sampling step at each timestep during training. Furthermore, both use a dataset of Bach chorales which are quantized music to quavers, disallowing shorter-duration notes such as semiquavers and demisemiquavers.

3.2 Automatic stylistic composition

While symbolic methods for automatic stylistic composition had been previously researched [27, 19], the rising popularity of connectionist methods coincided with a surge of models for automatically composing music ranging from baroque [63] to blues [36] to folk music [100]. This correlation is unsurprising: as connectionist models are trained to capture regularities in their training data, they are ideally suited for automatically composing music of a particular style.

3.2.1 Applications to Bach chorales

The Bach chorales have been a popular dataset for automatic composition research. Early systems primarily focused on chorale harmonization tasks and include rule-based systems leveraging hand-crafted rules [35] as well as models learned from data like the effective Boltzmann machine model [7]. Hybrids which learn rules for Bach from data have also been proposed [97].

An important work in automatic stylistic composition of Bach chorales is Allan and Williams [2], which applied harmonization HMM to generate harmonizations and a separate ornamentation HMM to fill in semiquavers. Their work is one of the first to quantitatively evaluate model performance using validation set cross-entropy and they introduce a dataset of Bach chorales (commonly referred to as *JSB Chorales* by other work). However, their harmonization HMM leverages a domain-specific harmonic encoding of chords for hidden states. Additionally, the dataset they introduced is quantized to quavers and hence affects all other models utilizing *JSB Chorales*.

The *JSB Chorales* introduced by Allan and Williams [2] has since become a standard evaluation benchmark routinely used [13, 87, 6, 49, 113] to evaluate the performance of sequence models on polyphonic music modelling. The current state-of-the-art on this dataset, as measured by cross-entropy loss on held-out validation data, is achieved by the RNN-DBN [49].

While the introduction of the standardized *JSB Chorales* dataset has helped improve performance evaluation, it does not solve the problem of measuring the success of an automatic stylistic composition. This is because the goal of an automatic stylistic composition system is to generate music which human evaluators find similar to a particular style, not to maximize cross entropy on unseen test data.

3.2.2 Evaluation of automatic composition systems

This difficulty in evaluating automatic composition systems was first addressed by Pearce and Wiggins [89]. Lack of rigorous evaluation affects many of the earlier automatic composition systems and complicates performance comparisons. Even with standard corpuses such as *JSB Chorales*, cross-entropy is still a proxy to the true measure of success for an automatic stylistic composition system.

In order to obtain a more direct measure of success, researchers have turned to subjective evaluation by human listeners. *Kulitta* [91] is a recent rule-based system whose performance was evaluated by 237 human participants from Amazon MTurk. However, their participant pool consists entirely of US citizens (a fault of MTurk in general) and the data obtained from MTurk is of questionable quality [34]. Moreover, their results only indicated that participants believed *Kulitta* to be closer to Bach than to a random walk. The use of a large pool of human evaluators represents a step in the right direction. However, a more diverse participant pool coupled with stronger results would significantly improve the strength of this work.

Perhaps most relevant to our work is *Racchmaninof* (RAndom Constrained CHain of MArkovian Nodes with INheritance Of Form) by Collins et al. [20], an expert system designed for stylistic automatic composition. The authors evaluate their system on 25 participants with a mean of 8.56 years of formal music training and impressively find that only 20% of participants performed significantly better than chance. While we believe this to be one of the most convincing studies on automatic stylistic composition to date, a few criticisms remain. First, the proposed model is highly specialized to automatic stylistic composition and is more of a testament to the author's ability to encode the stylistic rules of Bach than to the model's ability to learn from data. Additionally, a larger and more diverse participant group including evaluators of varying skill level would provide stronger evidence of the model's ability to produce "Bach-like" music to average human listeners.

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 [14]

4

Automatic stylistic composition with deep LSTM

This chapter describes the design and quantitative evaluation of a generative RNN sequence model for polyphonic music. In contrast to many prior systems for automatic composition, we intentionally avoid allowing our prior assumptions about music theory and structure impact the design of our model, opting to learn features from data over injecting prior knowledge. This choice is motivated by three considerations:

1. Prior assumptions about music may be incorrect, limiting the performance achievable by the model
2. The goal is to assess the model's ability to compose convincing music, not the researcher's prior knowledge
3. The structure learned by an assumption-free model may provide novel insights into various musical phenomena

Note that this is deviates from many prior works, which leveraged domain-specific knowledge such as modelling chords and notes hierarchically [58, 81, 38], accounting for meter [37], and detecting for motifs [41].

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 [90] (*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"'[83] which Bach delimited using fermatas

3. The Bach chorales have become a standardized corpus routinely studied by music theorists[110]

While the *JCB Chorales* [2] has become a popular dataset for polyphonic music modelling, we will show in 4.1.1 that its quantization to quavers introduces a non-negligible amount of distortion.

Instead, we opt build a corpus of Bach chorales which is quantized to semiquavers rather than quavers, enabling our model to operate at a **time resolution at least** $2\times$ **better than all related work**.

Our data is obtained from the **Bach-Werke-Verzeichnis** (BWV) [17] indexed collection of the Bach chorales provided by the `music21`[28] Python library.

4.1.1 Preprocessing

Motivated by music's transposition invariance (see section A.2.4 on page 76) as well as prior practice [81, 38, 43, 42], we first perform **key normalization**. The keys of each score were first analyzed using the Krumhansl Schmuckler key-finding algorithm [72] 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 fundamental duration. Our model uses a fundamental duration of one semibreve, **exceeding the time resolutions of [13, 38] by 2x, [58] by 4x, and [7] by 8x**.

We consider only note pitches and durations, neglecting changes in timing (*e.g.* ritardandos), dynamics (*e.g.* crescendos), and additional notation (*e.g.* accents, staccatos, legatos). This is comparable to prior work [13, 87] where a MIDI-encoding also lacking this additional notation was used.

An example of the effects introduced by our preprocessing 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 25.

Quantizing to semiquavers introduces non-negligible distortion

Choosing to implement our own sequential encoding scheme was a difficult choice. While it would permit a finer time-resolution of semiquavers, it would make our cross-entropy losses incomparable to those reported on *JCB Chorales* [2].

To justify our decision, we investigated the distortion introduced by quantization to quavers rather than semiquavers in fig. 4.3 on page 26. We find that *JCB Chorales* **distorts** 2816 **notes in the corpus** (2.85%) **because of quantization to quavers**. Since our research aim is to generate convincing music, we **minimize unnecessary distortions and proceed with our own encoding scheme**. We understand that this choice will create difficulties in evaluating our model's success, and address this concern through alternative means (chapter 7) of evaluation which are arguably more relevant for automatic stylistic composition systems.

We also investigate changes in other corpus-level statistics as a result of key normalization and time quantization, such as pitch and pitch class usages and meter. All results fall within expectations, but the interested reader is directed to section C.1.1 on page 82.

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.

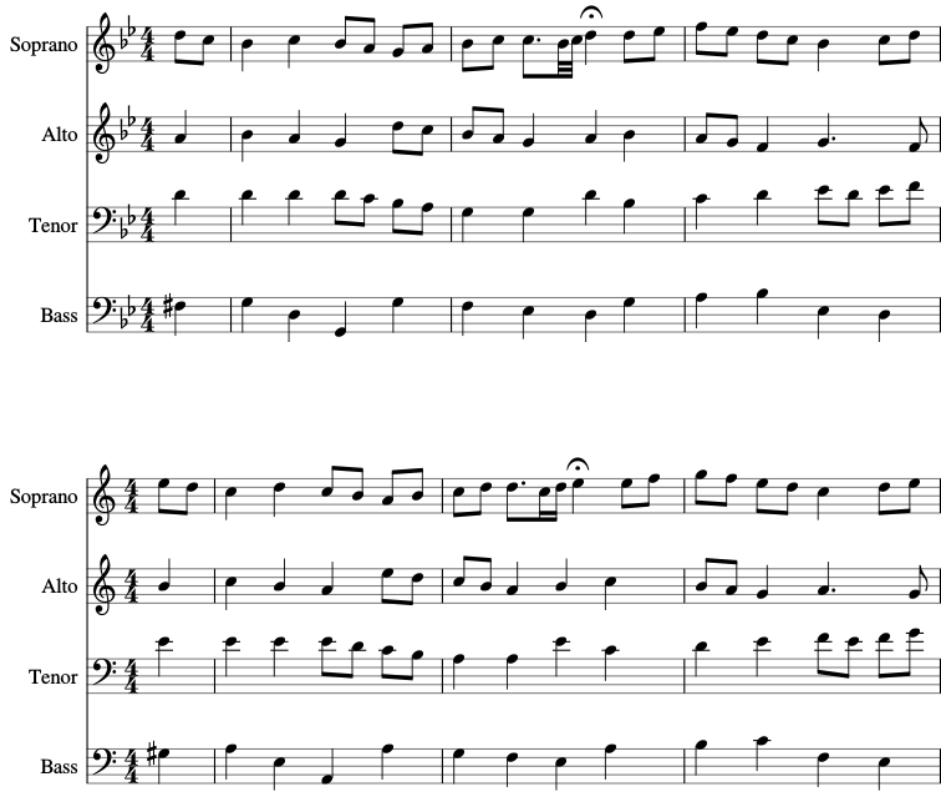


Fig. 4.1 First 4 bars of JCB Chorale BWV 185.6 before (top) and after (bottom) preprocessing. Note the transposition down by a semitone to C-major as well as quantization of the demisemiquavers in the third bar of the Soprano part.

Token-level versus frame-level encoding

One design decision is whether the tokens in the sequence are comprised of individual notes (as done in [81, 43, 99]) or larger harmonic units (*e.g.* chords [38, 13], "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. The use of **a note-level encoding may improve performance with respect to out-of-vocabulary (OOV) tokens** in two ways. It first reduces the potential vocabulary size from $O(128^4)$ possible chords down to $O(128)$ potential notes. In addition, harmonic relationships learned by the model parameters may enable generalization to OOV queries (*e.g.* OOV chords that are transpositions of in-vocabulary chords).

In fact, the decision may not even matter at all. Graves [52] 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 choice of notes versus chords is not very significant, at least for English language modelling.

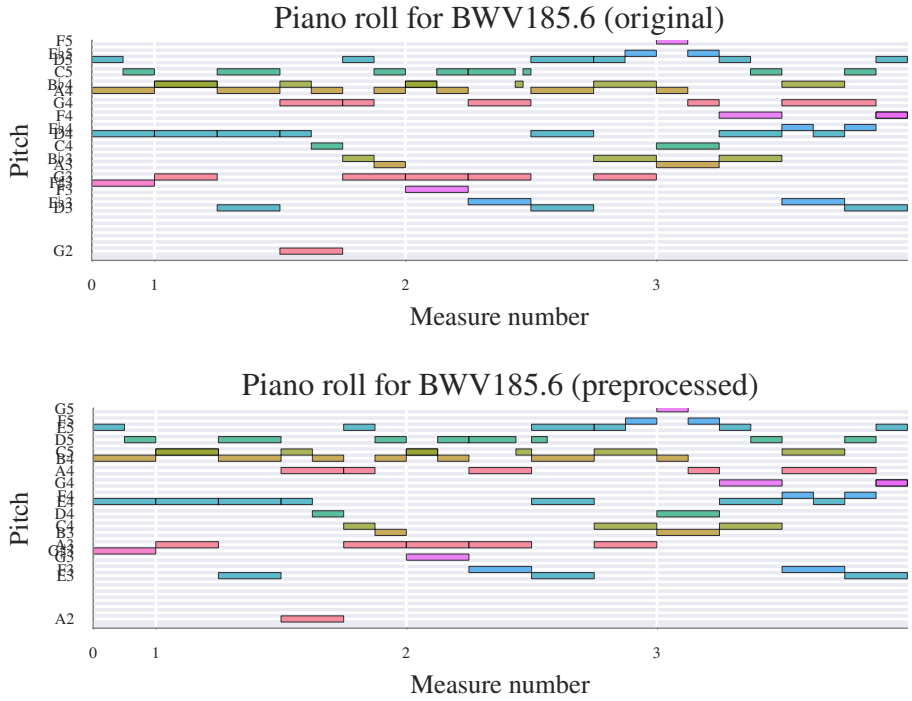


Fig. 4.2 Piano roll representation of the same 4 bars from fig. 4.1 before and after preprocessing. Again, note the transposition to C-major and time-quantization occurring in the Soprano part.

Definition of the encoding scheme

Similar to [105], our encoding represents 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 [48] demonstrated LSTM is capable of. Consecutive frames are separated by a unique delimiter ("|||"' in fig. 4.4 on page 27). 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 $\in \{True, False\}$ distinguishes whether a note is tied with a note at the same pitch from the previous frame or is articulated at the current timestep.

For each score, a unique start symbol ("START" in fig. 4.4) and end symbol ("END" in fig. 4.4) 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.

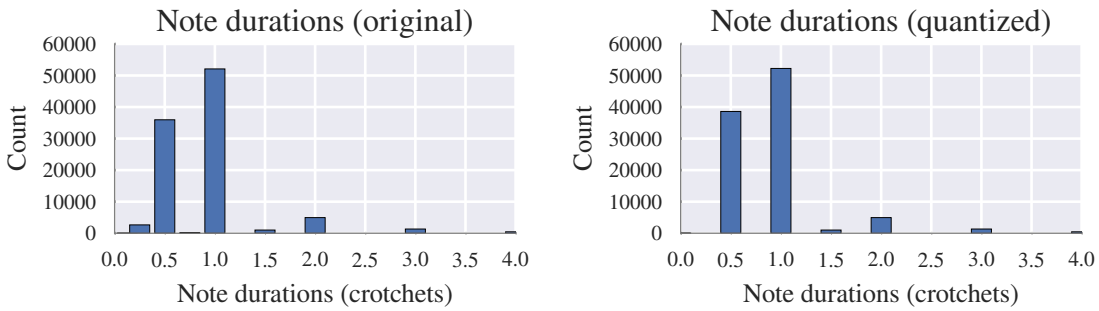


Fig. 4.3 Distortion introduced by quantization to semiquavers

Table 4.1 Statistics on the preprocessed datasets used throughout our study

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

.

Ordering of parts within a frame

A design decision 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 would like to process Soprano notes first and **order the notes within a frame in descending pitch**.

Modelling fermatas produces more realistic phrasing

The above specification describes our initial attempt at an encoding format. However, we found that this encoding format resulted in unrealistically long phrase lengths. **Including fermatas** (represented by "(.)" in fig. 4.4 on the facing page), which Bach used to denote ends of phrases, **helped alleviate problems with unrealistically long phrase lengths**.

Encoded corpus statistics

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. 4.5 and confirms the failure of Zipf's law in our data. This shows that our data's distribution differs from those typical for language corpuses, suggesting that the N-gram language models benchmarked in section 4.2.2 on page 29 may not perform well.

```
START
(59, True)
(56, True)
(52, True)
(47, True)
|||
(59, True)
(56, True)
(52, True)
(47, True)
|||
(.)
(57, False)
(52, False)
(48, False)
(45, False)
|||
(.)
(57, True)
(52, True)
(48, True)
(45, True)
|||
END
```

Fig. 4.4 Example encoding of a score containing two chords, both one quaver in duration and the second one possessing a fermata. Chords are encoded as (MIDI pitch value, tied to previous frame?) tuples, "|||"' encodes the ends of frames, and "(.)" at the start of a chord encodes a fermata. Each "|||" corresponds to time advancing by a semiquaver

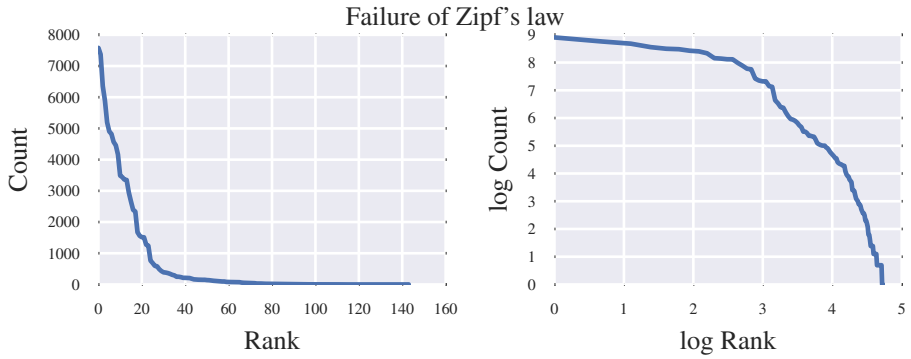


Fig. 4.5 Left: Token frequencies sorted by rank. Right: log-log plot where a power law distribution as predicted by Zipf's law would appear linear.

Discussion on our encoding scheme

We make the following observations about our proposed encoding scheme:

- It is **sparse**: unarticulated notes are not encoded
- It is also **variable length**: each frame can span anywhere from one to five tokens, requiring LSTM's capability of detecting spacing between events[48]
- The explicit representation of tied notes vs articulated notes **enables us to determine when notes end**, resolving an issue present in many prior works [38, 37, 75, 15]

Unlike many others [81, 43, 73], we avoid adding prior information through engineering harmonically relevant features. Instead, we appeal to results by Bengio [9] and Bengio and Delalleau [10] suggesting that that a key ingredient in deep learning's success is its **ability to learn good features from raw data**. Such features are very likely to be musically relevant, which we will explore further in chapter 5.

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.

4.2.1 Training and evaluation criteria

Following [81], we will train the model to predict $P(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$: a probability distribution over all possible next tokens $\boldsymbol{x}_{t+1}$ given the current token $\boldsymbol{x}_t$ and the previous hidden state

$\boldsymbol{h}_{t-1}$. This is the exact same operation performed by RNN language models [80]. We minimize cross-entropy loss between the predicted distributions $P(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$ and the actual target distribution $\delta_{\boldsymbol{x}_{t+1}}$

At the next timestep, the correct token $\boldsymbol{x}_{t+1}$ is provided 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** [112] performed to aid convergence because the model's predictions may not be reliable early in 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 [8] 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 [57] and SRILM [98], 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. 4.5 on page 28). Furthermore, both are based upon N-gram models which are constrained to only account for short-term dependencies. Therefore, we expect RNNs to outperform the N-gram baselines shown in table 4.2 on the next 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

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

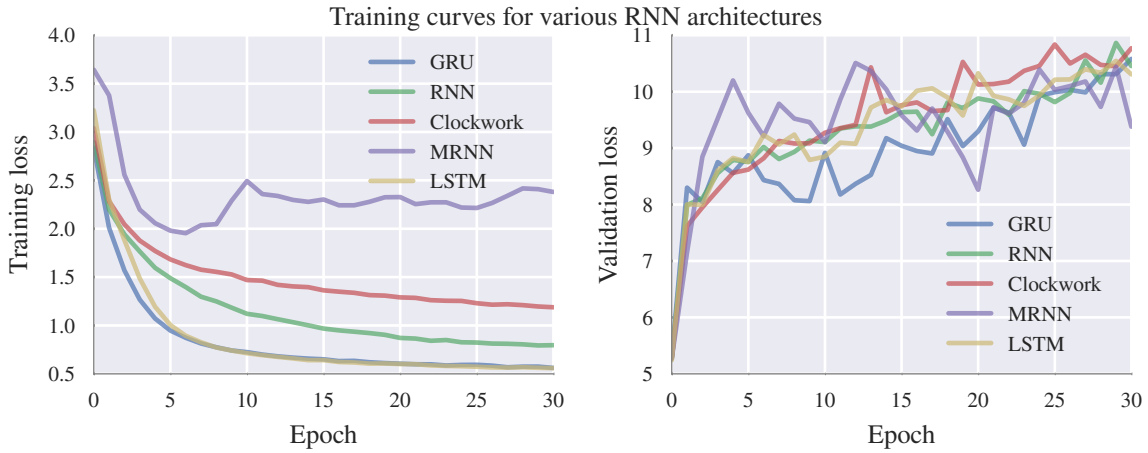


Fig. 4.6 LSTM and GRUs yield the lowest training loss. Validation loss traces show all architectures exhibit signs of significant overfitting

4. `seq_length` – number of frames before truncating BPTT gradient

5. `dropout` – the dropout probability

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 [65] and dropout [60] 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.

Models were trained using Adam [70] with an initial learning rate of 2×10^{-3} decayed by 0.5 every 5 epochs. The back-propagation through time gradients were clipped at ± 5.0 [86] and BPTT was truncated after `seq_length` frames. A minibatch size of 50 was used.

4.2.4 Comparison of memory cells on music data

We used `theanets`[1] to rapidly implement and compare a large number of memory cell implementations. Figure 4.6 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 configured the clockwork RNN [18] with 5 equal-sized hidden state blocks with update periods $(1, 2, 4, 8, 16)$.

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

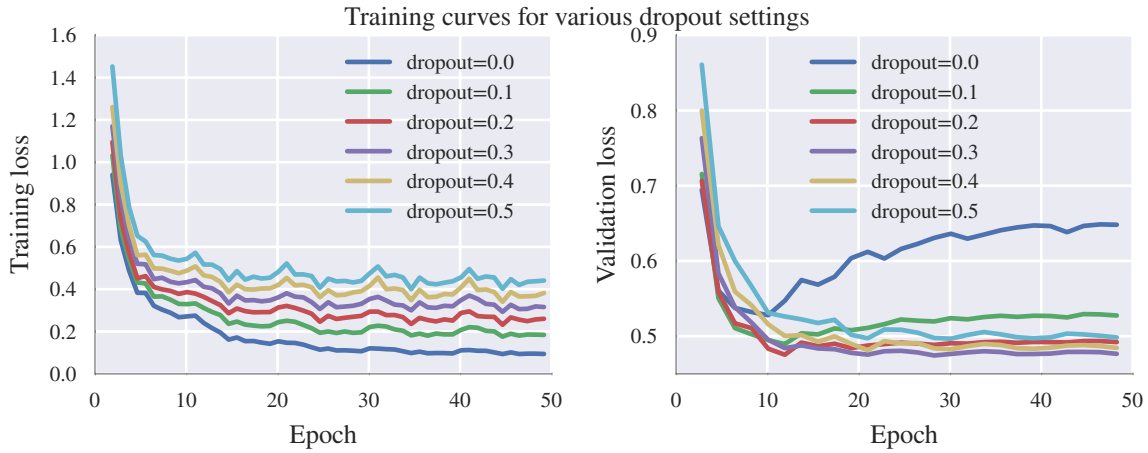


Fig. 4.7 Dropout acts as a regularizer, resulting in larger training loss but better generalization as evidenced by lower validation loss. A setting of `dropout=0.3` achieves best results for our model.

Figure 4.6 shows that while all models achieved similar validation losses, LSTM and GRUs trained much faster and achieved lower training loss. Since Zaremba [113] find similar empirical performance between LSTM and GRUs and Nayebi and Vitelli [84] 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. 4.6 is a red flag suggesting that overfitting is occurring. This observation motivates the exploration of dropout regularization in section 4.2.5.

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 (see table 4.3 on page 34), dropout, and batch normalization.

Dropout regularization improves validation loss

The increasing validation errors in fig. 4.6 on page 31 prompted investigation of regularization techniques. In addition to adding batch normalization, a technique known to reduce overfitting and accelerate training [65], we also investigated the effects of different levels of dropout by varying the `dropout` parameter.

The experimental results are shown in fig. 4.7. As expected, dropout acts as a regularizer and reduces validation loss from 0.65 down to 0.477 (when `dropout=0.3`). Training loss

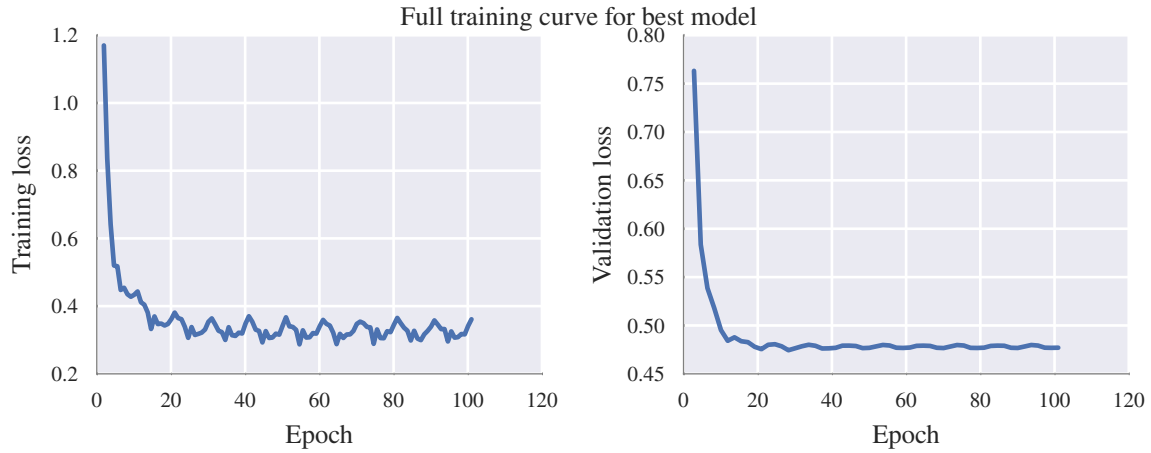


Fig. 4.8 Training curves for the overall best model. The periodic spikes correspond to resetting of the LSTM state at the end of a training epoch.

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` $\in \{1, 2, 3, 4\}$
- `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 fig. C.5 on page 84.

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 4.8 plots the training curve of this model and shows that **training converges after only 30 iterations** (≈ 28.5 **minutes on a single GPU**).

To confirm local optimality, we perform perturbations about our final hyperparameter settings in figs. C.6 to C.10. Our analysis of these experiments yield the following insights:

Table 4.3 Timing results comparing CPU and GPU training of the overall best model (section 4.2.5 on page 33)

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

1. Depth matters! Increasing `num_layers` can yield up to 9% lower validation loss. The best model is 3 layers deep, any further and overfitting occurs. This finding is unsurprising: the dominance of deep RNNs in polyphonic modelling was already noted by Pascanu et al. [87]

2. Increasing hidden state size (`rnn_size`) improves model capacity, but causing overfitting when too large

3. The exact size of the vector embeddings (`wordvec`) did not appear significant

4. While training losses did not change, increasing the BPTT truncation length (`seq_length`) decreased validation loss, suggesting improved generalization

4.2.6 GPU training yields 800% acceleration

Consistent with prior work [102, 68], timing results table 4.3 from training our overall best model confirmed a 800% speedup enabled by the GPU training implemented in `torch-rnn`.

4.3 Results and comparison

As done by [6, 13], we quantitatively evaluate our models using cross entropies and perplexities on a 10% held-out validation set. Our best model (fig. C.5 on page 84) achieves **cross-entropy losses of** 0.323 **on training data and** 0.477 **on held-out test data, corresponding to a training perplexity of** 1.251 **bits and a test perplexity of** 1.391. As expected, the deep LSTM model achieves more than 0.6 **bits lower than any validation perplexity obtained by the N-gram models** compared in table 4.2 on page 30.

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 [62]

5

Opening the black box: analyzing the learned music representation

A common criticism of deep learning methods are their lack of interpretability, an area where symbolic rule-based methods particularly excel. In this section, we argue the opposite viewpoint and demonstrate that characterization of the concepts learned by the model can be surprisingly insightful. The benefits of cautiously avoiding prior assumptions pay off as we discover the model itself learns musically meaningful concepts without any supervision.

5.1 Investigation of neuron activation responses to applied stimulus

Inspired by stimulus-response studies performed in neuroscience, we choose to characterize the internals of our sequence model by applying an analyzed music score as a stimulus and measuring the resulting neuron activations. Our aim is to see if any of the neurons have learned to specialize to detect musically meaningful concepts.

We use as stimulus the music score shown in fig. 5.1, which has already been preprocessed as described in section 4.1.1 on page 23. To aid in relating neuron activities back to music theory, chords are annotated with Roman numerals obtained using `music21`'s auto-

mated analysis. Note that Roman numeral analysis involves subjectivity, and the results of automated analyses should be carefully interpreted.

5.1.1 Pooling over frames

In order to align and compare the activation profiles with the original score, all the activations occurring in between two chord boundary delimiters must be combined. This aggregation of neuron activations from higher resolution (*e.g.* note-by-note) to lower resolution (*e.g.* frame-by-frame) is reminiscent of pooling operations in convolutional neural networks [94]. Motivated by this observation, we introduce a method for pooling an arbitrary number of token-level activations into a single frame-level activation.

Let $\boldsymbol{y}^{(l)}_{t_m:t_n}$ denote the **activations** (*e.g.* outputs) 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** $\tilde{\boldsymbol{y}}^{(l)}_n$ to be the element-wise maximum of $\boldsymbol{y}^{(l)}_{t_m:t_n}$, that is:

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

where $\boldsymbol{y}^{(l)}_{t,i}$ 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 $\tilde{\boldsymbol{y}}$ 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. This is note the case for unpooled token-level activations (see fig. C.4 on page 85).

Notice the vertical bands corresponding to when a chord/rest is held for multiple frames. Also, the vector embedding corresponding to (*e.g.* near frames 30 and 90 in fig. 5.2 top) are sparse, showing up as white smears on the LSTM memory cells at all levels of the model.

5.1.2 Probabilistic piano roll: likely variations of the stimulus

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 some arbitrary ordering of tokens within the vocabulary. To aid interpretation, the tokens can be mapped back to their corresponding

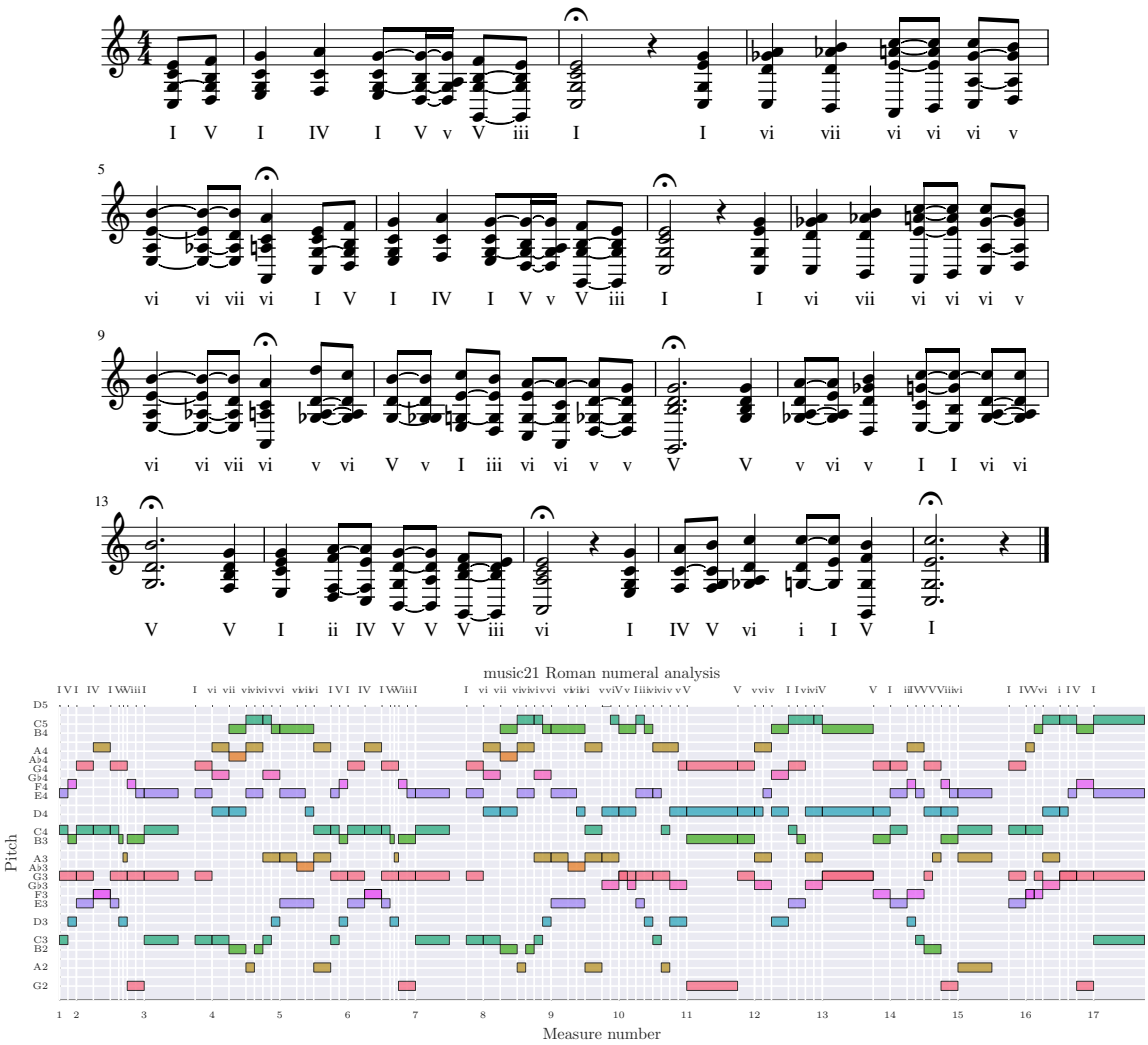


Fig. 5.1 *Top*: The preprocessed score (BWV 133.6) used as input stimulus with Roman numeral analysis annotations obtained from `music21`; *Bottom*: The same stimulus represented on a piano roll

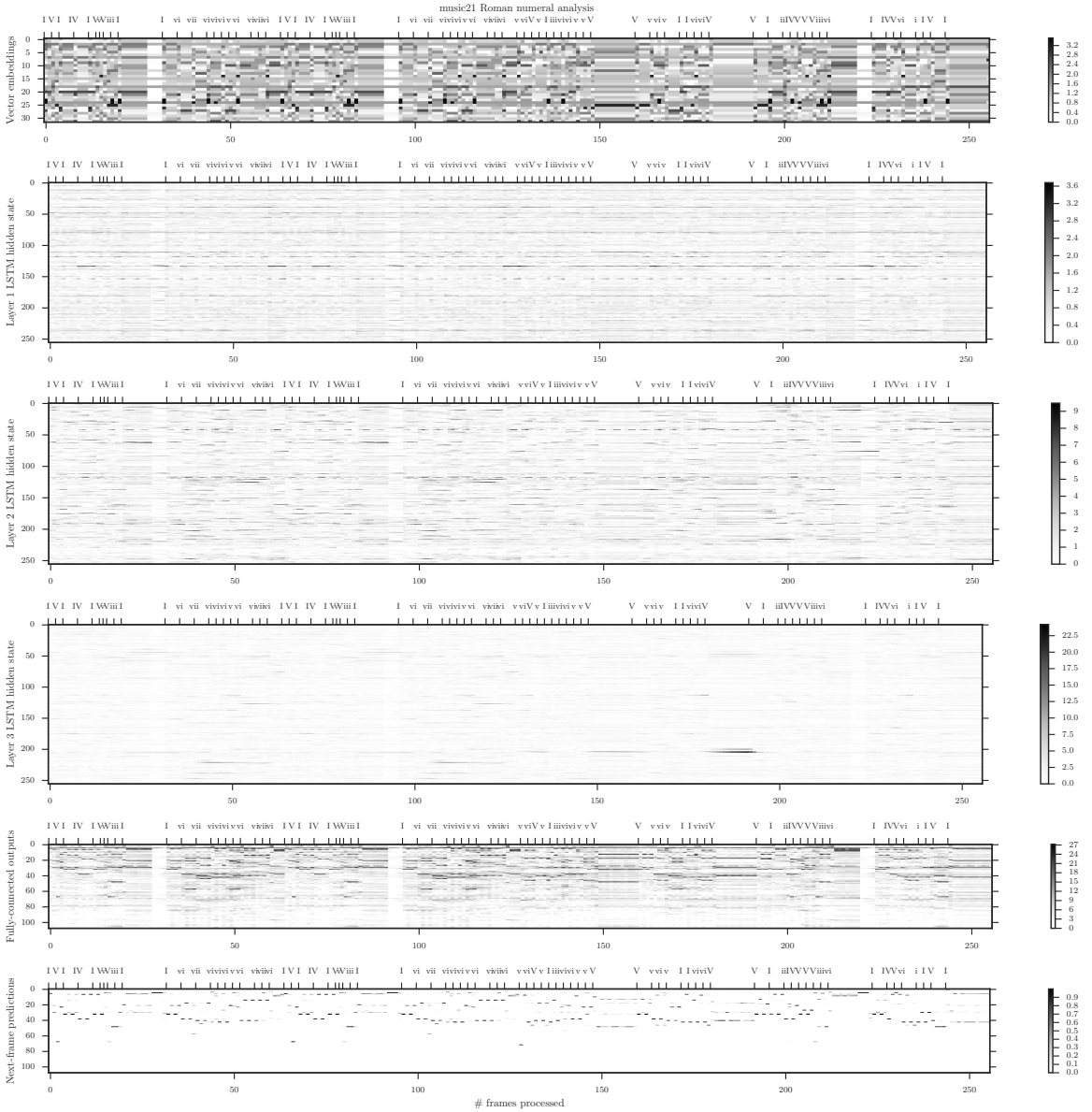


Fig. 5.2 Neuron activations after max pooling over frames

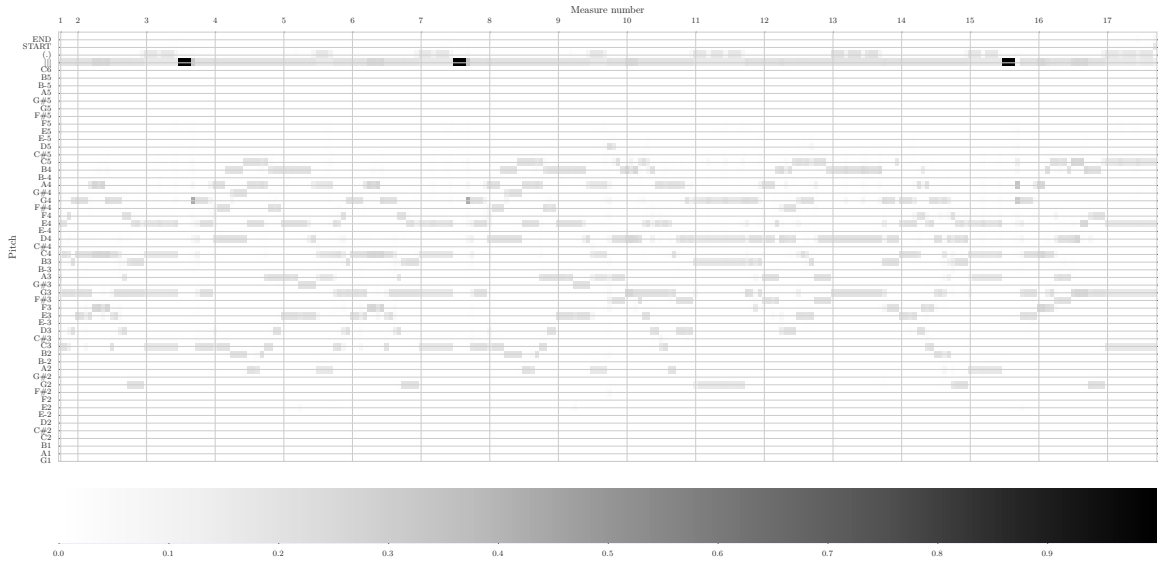


Fig. 5.3 Probabilistic piano roll of next note predictions. The model assigns high probability to fermatas near ends of phrases, suggesting an understanding of phrase structure in chorales.

pitches and laid out to reconstruct a **probabilistic piano roll**[37] consisting of the model's sequence of next-frame predictions as it processes the input. This is shown in fig. 5.3.

One surprising insight from fig. 5.3 is the high probability predictions for fermatas (third from top "(.)") near the end of phrases. This could indicate the model has managed to learned a notion of phrase structure.

Another interesting row of fig. 5.3 corresponds to frame delimiters (fourth from top, "|||"). Notice that the predictions for frame 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, suggesting that there are more permissible notes to play at the end of a rest than in the middle of a phrase. Finally, the probability assigned to fermatas is larger near the ends of phrases, suggesting that the model has learned some notion of of phrasing within music.

However, it may also not be very significant. Notice that the probabilistic piano roll in fig. 5.3 closely resembles the stimulus. This is because the recurrent inputs are taken from the stimulus rather than sampled from the model's predictions (a.k.a. [112]), 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.

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 [115]. 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. We discover certain neurons whose activities appear correlated to specific motifs, chord progression, and phrase structures, and show their activity profiles in fig. 5.4.

Given our limited knowledge of music theory, we provided the activation profiles for our collaborator Dr. Mark Gotham Gotham [51], who provided the following remarks:

- The first two (Layer 1, Neuron 64 / Layer 1, Neuron 138) seem to pick out (specifically) perfect cadence with root position chords in the the tonic key.
- There are no imperfect cadences here; just one interruptions into bar 14.
- Layer 1, Neuron 87: the I^6 chords on the first downbeat, and its reprise 4 bars later.
- Layer 1, Neuron 151: the two equivalent a minor (originally b minor) cadences that end phrases 2 and 4.
- Layer 2, Neuron 37: Seems to be looking for I^6 chords: strong peak for a full I^6; weaker for other similar chords (same bass).
- The rest are less clear to me.

Dr. Gotham's analysis suggests that while some neurons are ambiguous to interpretation, other neurons have learned highly specific and musically relevant concepts.

To our knowledge, this is the first reported result demonstrating LSTM neurons specializing to detect musically meaningful features. As we were careful to avoid imposing prior assumptions when designing the model, these neurons learned to specialize as a result of exposure to the Bach data. While we are hesitant to make broader conclusions from this single experiment, the implications of this finding are tremendously exciting for music theorists and deep learning researchers alike. We propose future work in this area in section 8.3 on page 59.

Fig. 5.4 Activation profiles demonstrating that neurons have specialized to become highly specific detectors of musically relevant features

6

Chorale harmonization

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.

Mirroring the pedagogy for music students, this chapter evaluates the learned deep LSTM model's ability on various harmonization tasks. Unlike automatic composition, where the model is free compose a score of music without any constraints, in harmonization tasks one or more of the parts are fixed and only the remaining are generated.

In music education, harmonization of a given melody is considered a more elementary task than generation of a novel chorale [30, 90]. However, these expectations may not be valid for our model, which was trained without any consideration of the future notes occurring in the provided melody. Our experiments in this chapter will yield a definitive answer to this question.

6.1 Adapting the automatic composition model

Recall that chapter 5 gave us a RNN model $\tilde{P}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$, which combined with a fixed initial hidden state $\boldsymbol{h}_0$ yields a **sequential prediction model** $\tilde{P}(\boldsymbol{x}_t|\boldsymbol{x}_{1:t-1})$ approximating the

true distribution $P^*(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)$. In this section, we describe a method for applying such a sequential prediction model to produce chorale harmonizations. The proposed technique is equivalent to a 1-best greedy search through a lattice constrained by the fixed melody line, and we will explore how solutions from the lattice processing literature might be applied.

In chorale harmonization, we are tasked to compose the notes for a subset of parts which are harmonically and stylistically compatible with the fixed parts. To be concrete, let $\{\boldsymbol{x}_t\}_{t=1}^T$ be a **sequence of tokens** representing an encoded score. Let $\alpha \subset \{1, 2, \cdots, T\}$ be **a multi-index** and suppose $\hat{\boldsymbol{x}}_\alpha$ correspond to the **fixed token values for the given parts** in the harmonization task. We are interested in optimizing the conditional distribution:

$$\boldsymbol{x}^*_{1:T} = \underset{x_{1:L}}{\operatorname{argmax}}\, P(\boldsymbol{x}_{1:L}|\boldsymbol{x}_\alpha = \hat{\boldsymbol{x}}_\alpha) \tag{6.1}$$

First, any **proposed solution** $\tilde{\boldsymbol{x}}_{1:T}$ must satisfy $\tilde{\boldsymbol{x}}_\alpha = \hat{\boldsymbol{x}}_\alpha$, so the only decision variables are $\tilde{\boldsymbol{x}}_{\alpha^c}$ where $\alpha^c \coloneqq \{1, 2, \cdots, T\} \setminus \alpha$. Hinton and Sejnowski [59] refer to this constraint as "clamping" the generative model.

The remaining task is to choose values for $\tilde{\boldsymbol{x}}_{\alpha^c}$. We propose a simple greedy strategy:

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

where the tilde on the previous tokens $\tilde{\boldsymbol{x}}_{1:t-1}$ indicate that they are equal to the actual previous argmax choices.

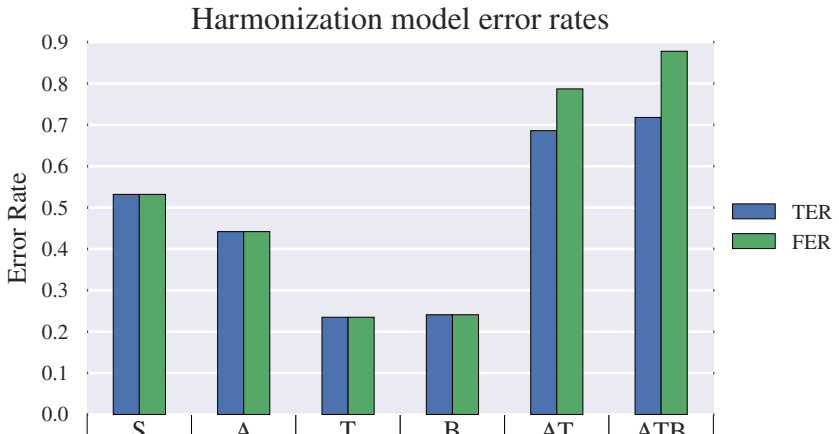
6.1.1 Shortcomings of the proposed model

We admit that this approach is a greedy approximation: we are not guaranteed $\hat{\boldsymbol{x}}_{1:T}$. In fact, we can view the problem as a search over a lattice of possible trajectories $\tilde{\boldsymbol{x}}_{1:T}$ constrained such that $\tilde{\boldsymbol{x}}_\alpha = \hat{\boldsymbol{x}}_\alpha$. Under the lattice framework, our strategy is recognized as a beam search with beam width 1. A wider beam-width which maintained N-best hypotheses such as in Liu et al. [76] would allow the model to partially recover from mistakes made by greedy action selection. We leave this extension for future work.

6.2 Datasets

We create datasets where one or more parts are to be harmonized by the model:

1. A single parts: Soprano (S), Alto (A), Tenor (T), or Bass (B).



	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 Token error rates (TER) and frame error rates (FER) for various harmonization tasks

2. The middle two parts (AT).

3. All parts except Soprano (ATB). This is what is usually meant by **harmonization**.

It is widely accepted that these tasks successively increase in terms of difficulty [30]. Of particular interest is the AT case. Bach oftentimes only wrote the Soprano and Bass parts of a piece, leaving the middle parts to be filled in by students. Based on these observations, we hypothesize that performance will be similar for harmonizing any single part and successively deteriorate for the AT and ATB cases respectively.

6.3 Results

6.3.1 Error rates harmonizing Bach chorales

We deleted different subsets of parts from a held-out validation corpus consisting of 10% of the data and used the method proposed in eq. (6.2) to harmonize the missing parts. We evaluate our model's error rate predicting individual tokens (**token error rate**, TER) as well as all tokens within frames (**frame error rate**, FER) and show our results in fig. 6.1.

Our results are surprising. Contrary to expectations, we found that error rates were significantly higher for S and A than for T and B. One possible explanation for this result is our design decision in section 4.1.2 on page 26 to order the notes within a frame in SATB order. While the model must immediately predict the Soprano part without any knowledge of the other parts, it has already processed all of the other parts and can utilize information about

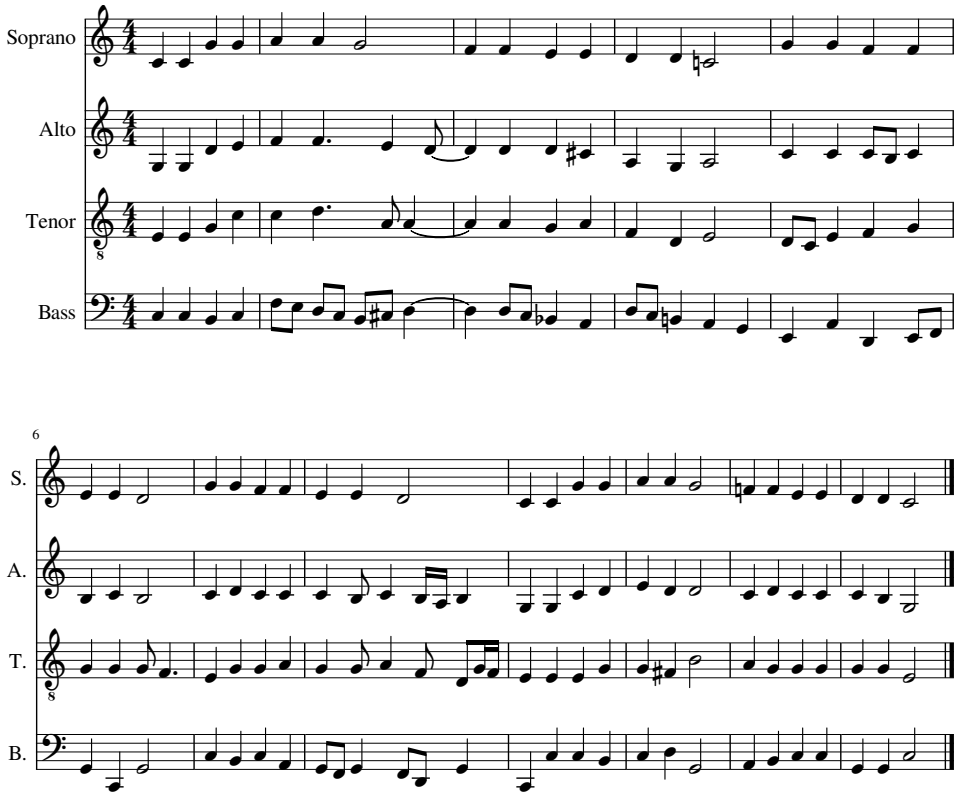


Fig. 6.2 BachBot's ATB harmonization to a *Twinkle Twinkle Little Star* melody

harmonic context when predicting the Bass parts. To further validate this idea, one could investigate trying other orderings in the encoding, which we propose as extension work.

6.3.2 Harmonizing popular tunes with BachBot

One question we would like to investigate is the generality of the concepts extracted by the model. Although it is trained for sequential prediction of Bach chorales, chapter 5 demonstrated that the model learns high-level music theoretic concepts. We hypothesized that these high-level concepts may enable the model to generalize to data which may significantly differ from its training data.

To test this hypothesis, we used BachBot to produce a harmonization for *Twinkle Twinkle Little Star*, a popular English lullaby published more than 50 years after Bach's era.

To our surprise, we found that BachBot was not only able to generate a harmonically pleasant harmonization, but that **the harmonization exhibited features reminiscent of Bach's Baroque style**. This result demonstrates that BachBot has successfully extracted statistical regularities from its input corpus which are involved in giving Baroque music its sense of style.

7

Large-scale subjective human evaluation

Evaluation of automatic composition systems is still a difficult question with no generally-accepted solution. While many recent models use log-likelihood on the *JCB Chorales* [2] as as a benchmark for comparing performance [13, 6, 87, 49, 113, 77], this evaluation merely measures a model's ability to approximate a probability distribution given limited data and does not correspond with success as an automatic composition problem.

Pearce and Wiggins [89] and Pearce, Meredith, and Wiggins [88] attribute difficulty in evaluation due to lack aim, claiming that studies in automatic music generation do not clearly define their goals. They proceed to differentiate between three different goals for automatic music generation research, each with different evaluation criteria:

1. Automatic composition
2. Design of compositional tools
3. Computational modelling of music style/cognition

While our model design and analysis has happened to yield interesting results relating to music cognition (chapter 5), it has not been the aim of our work. As initially stated in chapter 1, **the aim of our work is automatic composition**: to produce a system which automatically composes music indistinguishable from Bach.

To directly measure our success on this task, we adapt Alan Turing's proposed "Imitation Game" [108] into a musical Turing test. Although some authors [4] are critical of such tests' ability to provide meaningful data which can be utilized to improve the system, their recommended alternative of listener studies with music experts is cost-prohibitive and generates a small sample of free-form text responses which must be manually analyzed.

7.1 Evaluation framework design

In this section, we describe the design of a software framework for conducting large-scale a musical Turing test which was deployed to http://bachbot.com/ and used to evaluate our model on human evaluators.

7.1.1 Software architecture

We architected the evaluation framework with two requirements in mind:

1. It must scale in a cost-efficient manner to meet rapid growth
2. It must be easily adaptable for usage by others

Our scalability requirement motivated our choice for using a cloud service provider to manage infrastructure. While multiple options for providers, we chose to use Microsoft Azure. Our application is built using Node.js[1] and is hosted by Azure App Services. We host static content (*e.g.* audio samples) using Azure CDN to offload bandwidth. Responses collected from the survey are stored in Azure BlobStore, a distributed object store supporting batch MapReduce processing using Hadoop on HDInsight.

The frontend is built using React[2] and Redux[3]. We chose these frameworks because their current popularity in front-end web development implies familiarity by a large number of users, achieving our second software requirement. Additionally, Redux enables fine-grained instrumentation of user actions and allows us to collect detailed data such as when users play/pause the sample.

7.1.2 User interface

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

[1] https://nodejs.org/en/
[2] https://facebook.github.io/react/
[3] http://redux.js.org/

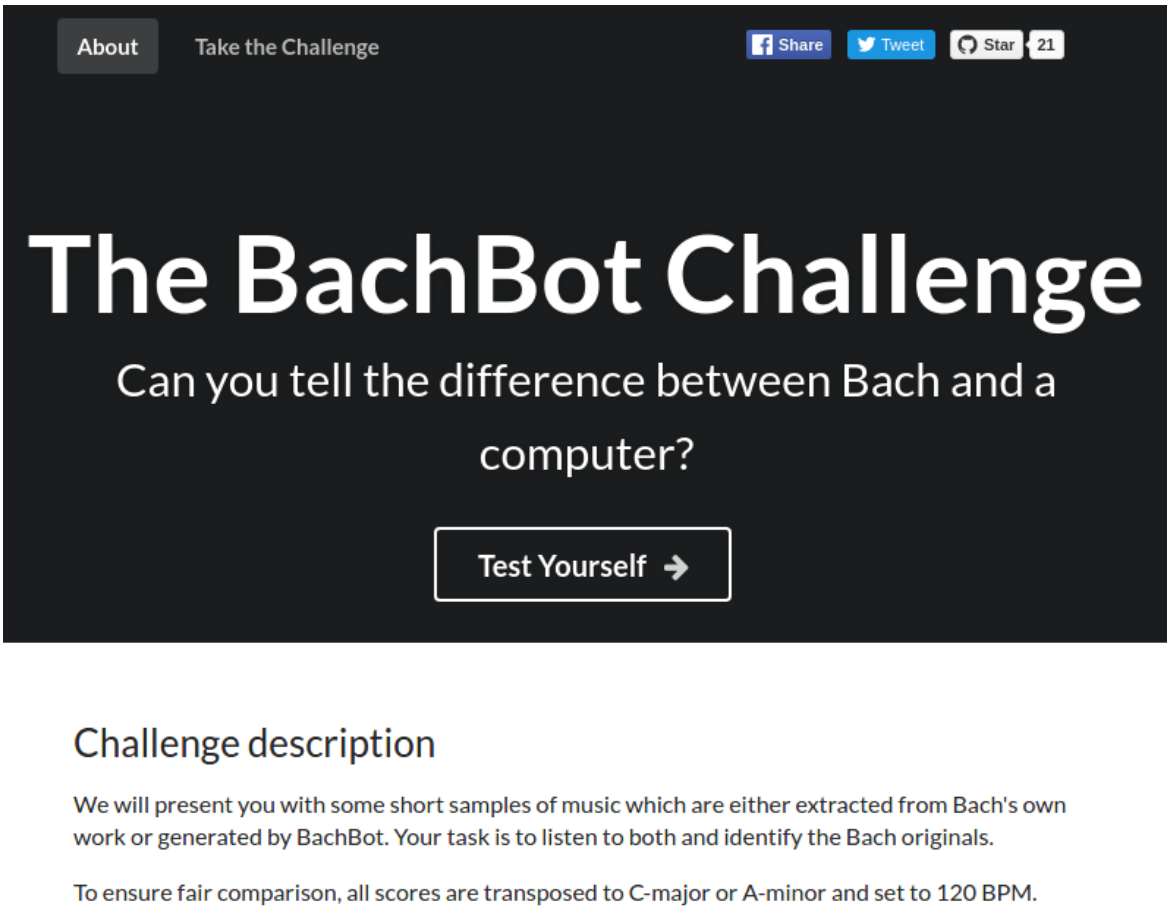


Fig. 7.1 The first page seen by a visitor of http://bachbot.com

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.

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 produced using the method proposed in chapter 6 as well as automatic compositions generated by sequentially sampling the model from chapter 4. We re-use notation from section 6.2 and use "SATB" to denote unconstrained automatic composition.

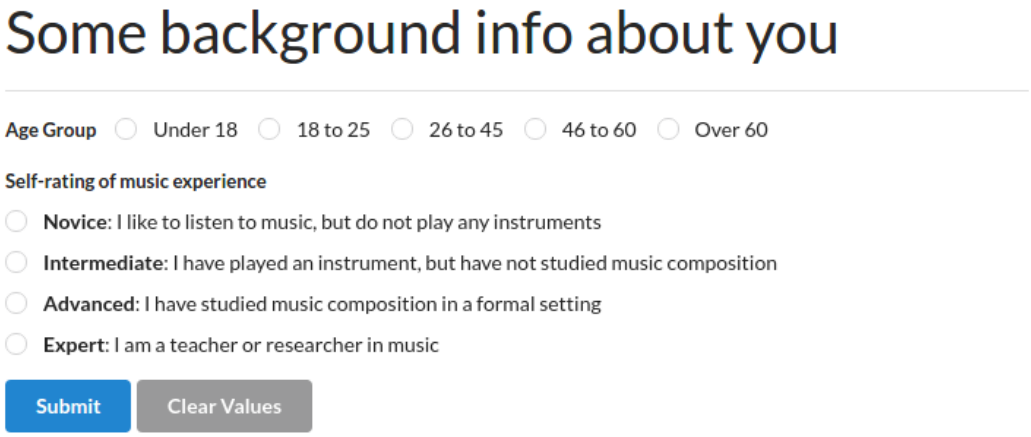


Fig. 7.2 User information form presented after clicking "Test Yourself"

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

For each question, a random chorale was selected without replacement from the corpus and paired with a corresponding harmonization. SATB samples 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

Unlike prior studies which leverage paid services like Amazon MTurk for human feedback [91], we do not not compensate participants and promote our study only through social media and personal contacts. Participation was voluntary and growth was organic; we did not solicit any paid responses or advertising.

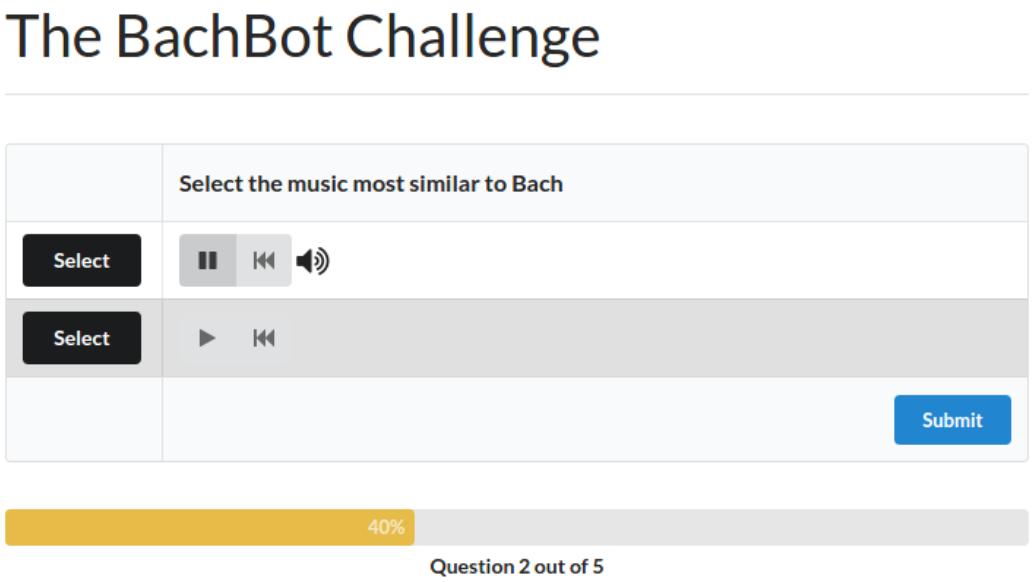


Fig. 7.3 Question response interface used for all questions

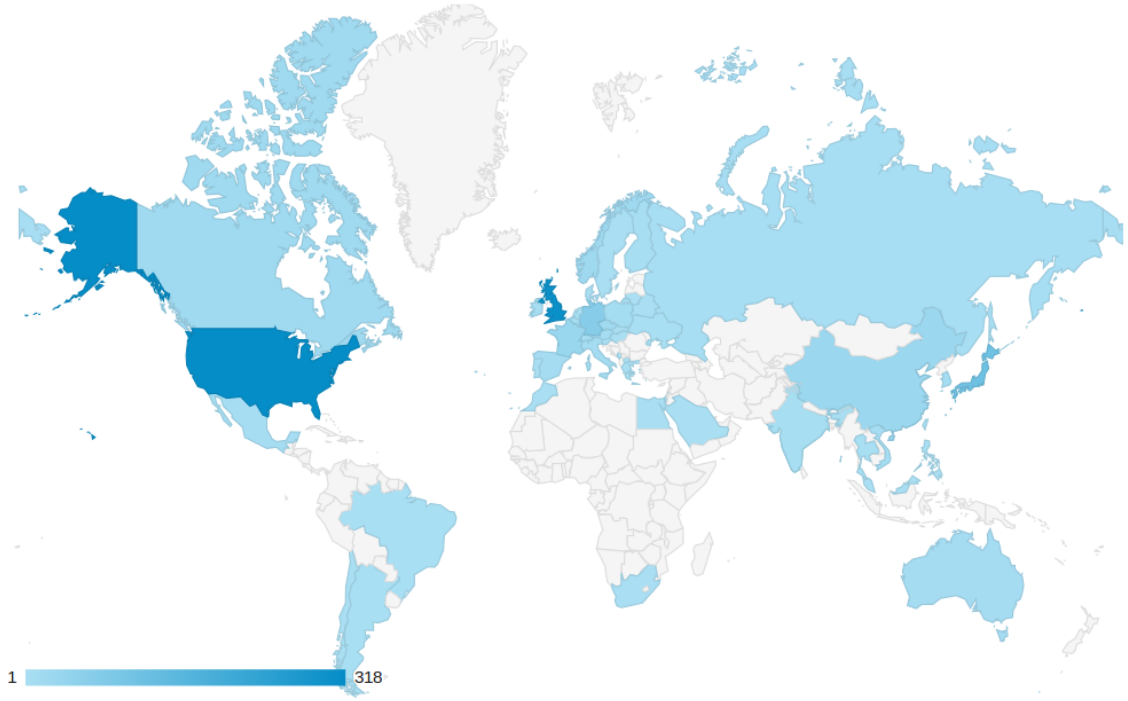
We found that 50% of participants were referred from social media, 4.8% through other websites and blogs, 2.6% through search engine results, and the remaining 42.6% had directly accessed bachbot.com,

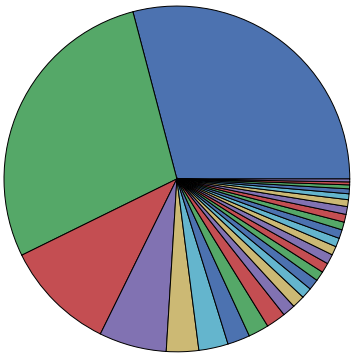
7.2 Results

7.2.1 Participant backgrounds and demographics

At the time of writing, we received a total of 759 participants from 54 different countries. After selecting only the first response per IP address and filtering down to participants whom had played both choices in every question at least once, we are left with 721 participants answering 5 questions each to yield a total of 3605 responses.

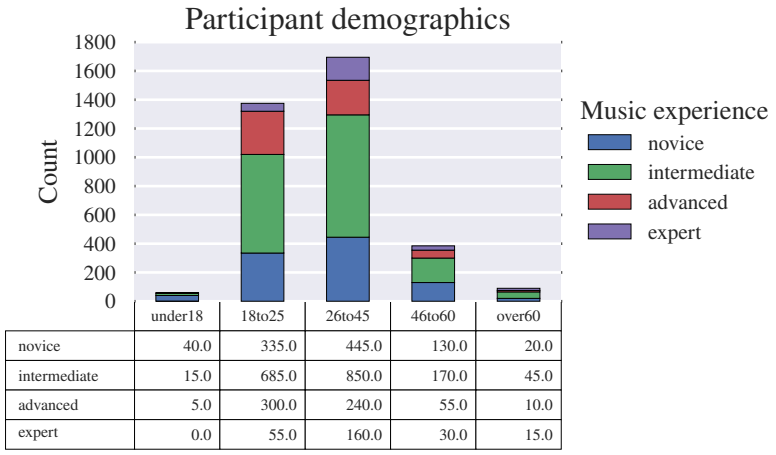
As evidenced by fig. 7.4 on the next page, our participant is diverse and includes participants from six different continents. fig. 7.5 on page 53 shows that the majority of our participants are between $18-45$ and have played an instrument. The large number of participants with significant music experience is larger than our expectations: more than 24.13% of participants have either formally studied or taught music theory. This large proportion of advanced participants shows that voluntary studies promoted through social media can generate significant participation by expert users interested in the problem domain.





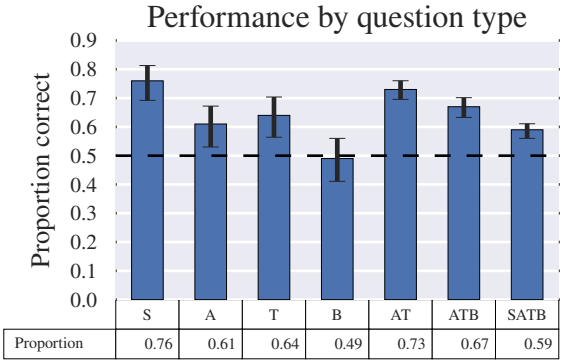
Country	Sessions	% All Sessions
United States	318	28.0
United Kingdom	308	27.0
Japan	114	10.0
Germany	69	6.0
France	33	3.0
China	30	3.0
Norway	23	2.0
Czech Republic	21	2.0
Canada	20	2.0
Italy	13	1.0
Poland	13	1.0
Australia	11	1.0
Spain	11	1.0
Switzerland	10	1.0
South Korea	10	1.0
Luxembourg	10	1.0
Austria	9	1.0
Belgium	9	1.0
Hong Kong	9	1.0
Ireland	8	1.0
Mexico	8	1.0
Vietnam	8	1.0
(not set)	64	1.0

Fig. 7.4 Geographic distribution of participants



	under18	18to25	26to45	46to60	over60
novice	40.0	335.0	445.0	130.0	20.0
intermediate	15.0	685.0	850.0	170.0	45.0
advanced	5.0	300.0	240.0	55.0	10.0
expert	0.0	55.0	160.0	30.0	15.0

Fig. 7.5 Demographics of participants



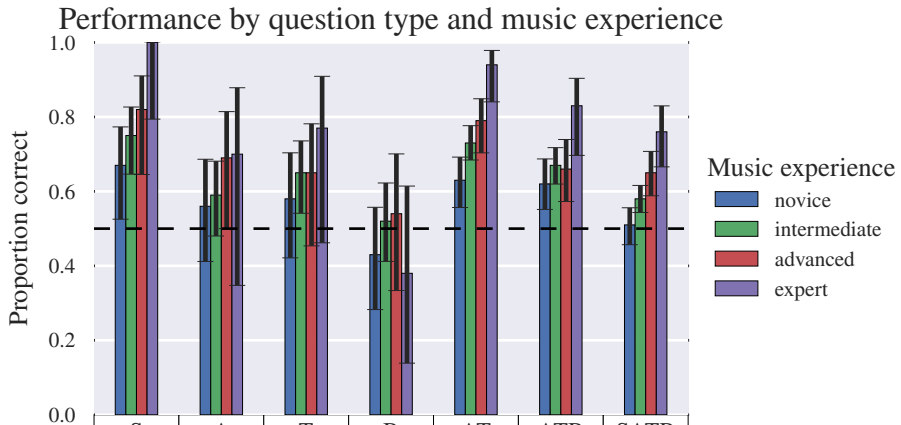
	S	A	T	B	AT	ATB	SATB
Proportion	0.76	0.61	0.64	0.49	0.73	0.67	0.59

Fig. 7.6 Proportion of participants correctly discriminating Bach from BachBot for each question type.

7.2.2 BachBot's performance results

Figure 7.6 shows the performance of BachBot on various question types. The SATB column shows that participants on average successfully discriminated Bach from BachBot only 59% of the time. As the baseline method of randomly guessing between the two choices in fig. 7.3 achieves 50%, our findings suggest that **the pool of participants on average could only perform** 9% **better than random guessing**. This is a much lower number than we anticipated and provides strong evidence affirming successful accomplishment of our research goals.

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



	S	A	T	B	AT	ATB	SATB
novice	0.67	0.56	0.58	0.43	0.63	0.62	0.51
intermediate	0.75	0.59	0.65	0.52	0.73	0.67	0.58
advanced	0.82	0.69	0.65	0.54	0.79	0.66	0.65
expert	1.0	0.7	0.77	0.38	0.94	0.83	0.76

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

The weaker performance of BachBot's outputs on harmonization questions (fig. 7.6, AT, ATB) compared to automatic composition questions (fig. 7.6, SATB) is counterintuitive as one might expect the provided parts to better aid the model to create more Bach-like music. However, this result is expected and can be explained by the shortcomings of our greedy 1-best harmonization method noted in section 6.1.1 on page 44.

BachBot's results compare favourably against a recently published comparable automatic stylistic composition system [20], which found that 5 out of 25 participants were able to significantly differentiate their generated compositions from Bach. However, the comparison must be interpreted with case as Collins et al. [20] evaluate on a highly experienced participant pool averaging 8.56.

When only a single part is composed by BachBot, we find the results vary significantly across different parts. Composing only 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 [30]. Composing only the Alto or Tenor parts achieved similar performance as composing all four parts.

However, using BachBot to compose new bass parts for Bach chorales yielded surprisingly successful results. Figure 7.6 shows that participants could not correctly discriminate Bach from BachBot any better than random guessing, and fig. 7.7 showed that even advanced and expert participants could not distinguish the two. This may be due to BachBot's lower error rates harmonizing Bass parts compared to any other part (fig. 6.1 on page 45), or it could also suggest that Bass parts are least significant in defining Bach's compositional style.

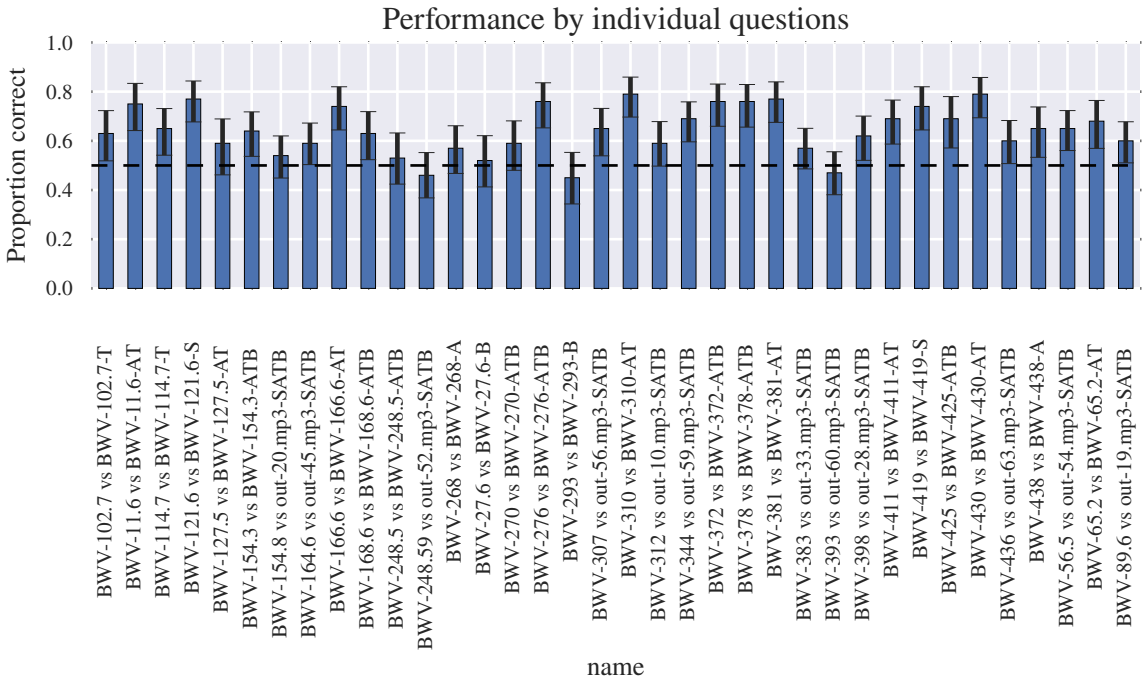


Fig. 7.8 Proportion of correct responses broken down by individual questions.

Certain compositions by BachBot cannot be significantly discriminated Bach

fig. 7.8 shows the proportion correct of correct responses received for each question. Encouragingly, it shows that more than a quarter of the SATB pairs were not insignificantly different from baseline, suggesting that **while not always consistent BachBot is capable of composing music which the average participant cannot discern from actual Bach**. While we lack the expertise, one valuable extension would be to review the samples which performed abnormally well and poor and attempt to identify some regularities among them.

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 [12]

8

Discussion, Conclusions, and Future Work

8.1 Discussion

From the data generated by bachbot.com, we are comfortable claiming that we have successfully accomplished our stated research aim of building an automatic stylistic composition system indistinguishable from actual Bach. In the process, we paid conscious effort to avoid allowing prior assumptions influence model design. By doing so, the performance of our model is a reflection of its ability to acquire music knowledge from data rather than the validity of the researcher's prior assumptions.

As a result of using our own music encoding scheme, we were unable to compare quantitative performance metrics such as log likelihood against many literature values reported for polyphonic modelling on the *JCB Chorales* [2]dataset. While this makes it difficult to evaluate our success, we affirm that the benefits justify the cost. The goal of our research is automatic stylistic composition, not probability density modelling. Hence, we prioritize producing realistic outputs and avoid the quantization distortion introduced by using *JCB Chorales*, opting instead to evaluate by other means.

Using this sequential encoding scheme, we train a deep LSTM sequential prediction model and discover that it learns music theoretic concepts without prior knowledge or explicit supervision. We then propose a method to utilize the sequential prediction model for

harmonization tasks. We acknowledge that our method is not ideal and discuss better alternatives in future work. Our harmonization results reveal that this issue is significant and should be a priority for any follow-up work.

Finally, we leveraged our model to generate harmonizations as well as novel compositions and used the generated music in a large-scale music Turing test. Our results here confirm the success of our project.

8.2 Summary of contributions

In this thesis, we make the following contributions:

- We introduce sequential encoding scheme for music achieving time-resolution 2× that of the commonly used *JCB Chorales* [2] dataset
- We demonstrate that a deep LSTM sequential prediction model trained on our encoding scheme is capable of composing music less than 9% of average listeners can reliably distinguish
- Our analysis of the neuron sensitivities within the learned LSTM model reveal that common music-theoretic concepts are acquired by the model without prior knowledge or supervision, a phenomena which to our knowledge has not been reported previously
- Performed the largest (to the best of our knowledge as of Friday 12$^{\text{th}}$ August, 2016) musical Turing test of an automatic composition system, which demonstrated that quality data can be collected from voluntary internet surveys

In addition, we have open sourced the code for BachBot[1] as well as our large-scale music Turing test framework[2]. These projects have all been received with excitement by the open source community and plans are in place to transfer the BachBot model to Google Magenta, a team within TensorFlow [1] which is focused on machine learning applications within music.

[1] https://github.com/feynmanliang/bachbot

[2] https://github.com/feynmanliang/subjective-evaluation-server and https://github.com/feynmanliang/subjective-evaluation-client

8.3 Extensions and Future Work

8.3.1 Improving harmonization performance

One significant opportunity for improvement is the harmonization method, which currently yields less than impressive results (section 6.3). The problem is that the sequential model is applied greedily with no accounting of the future constraints present in harmonization tasks. Our current approach suffers because a 1-best greedy approach may make a locally-optimal choice which severely impacts the likelihoods assigned by the model to the constrained future token outputs.

A potential solution to address this is to apply bidirectional RNNs[53] and the sequence to sequence framework[102] 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 [5] 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 view the problem in a lattice-based framework. Here, each valid harmonization corresponds to a path through a lattice constrained by the fixed parts to harmonize against. Under this lattice based interpretation, our strategy is recognized as a beam search with beam width 1. A wider beam-width which maintains N-best hypotheses such as in Liu et al. [76] would allow the model to partially recover from mistakes made by greedy action selection. Alternatively, a look-ahead search [85] extending to the next fixed token could also be used to help account for future constraints.

8.3.2 Ordering of parts in sequential encoding

We also found that the performance of harmonizing a single voice was strongly correlated with the order in which parts within the same frame were encoded (see section 4.1.2 on page 26). This suggests that the encoding scheme may have a more significant impact on model performance than we had imagined. 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.

8.3.3 Extensions to other styles and datasets

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 46, 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 generality and its failure modes by applying the model to other styles of music.

8.3.4 Analyzing results using music theory

At many points in our work, additional experience in music theory may have yielded insights not otherwise attainable. One instance of this is in fig. 7.8 on page 55, where 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 last future work we believe to be particularly exciting is to extend section 5.1.3 on page 40 and perform a statistical analysis of how closely the model neurons relate to input musical ideas. For instance, the input could be fixed to some known feature (*e.g.* all perfect cadences in the tonic key) and the neurons which fire could be examined. Alternatively, existing understanding can be validated by annotation regions of a score exhibiting a particular musical quality and searching for neurons with correlated activity.

References

[1] Martın Abadi et al. “Tensorflow: Large-scale machine learning on heterogeneous distributed systems”. In: **arXiv preprint arXiv:1603.04467** (2016).

[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] 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.

[5] 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.

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

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

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

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

[19] 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.

[20] 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.

[21] 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).

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

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

[24] David Cope. “Experiments in Music Intelligence”. In: **Proceedings of the International Computer Music Conference**. 1987.

[25] 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.

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

[27] 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.

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

[29] 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.

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

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

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

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

[34] 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.

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

[36] 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.

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

[38] 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.

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

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

[41] 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.

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

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

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

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

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] Mark Gotham. personal communication. Aug. 9, 2016.

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

[53] 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.

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

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

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

[57] 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.

[58] 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.

[59] 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.

[60] Geoffrey E Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: **arXiv preprint arXiv:1207.0580** (2012).

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

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

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

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

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

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

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

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

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

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

[71] 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.

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

[73] 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.

[74] 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.

[75] 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.

[76] 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.

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

[78] 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.

[79] 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.

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

[81] 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.

[82] Dylan Jeremy Nagler. “SCHUBOT: Machine Learning Tools for the Automated Analysis of Schubert’s Lieder”. PhD thesis. 2014.

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

[84] 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.

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

[86] 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.

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

[88] 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.

[89] 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.

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

[91] Donya Quick. “Kulitta: A Framework for Automated Music Composition”. PhD thesis. YALE UNIVERSITY, 2014.

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

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

[94] Dominik Scherer, Andreas Müller, and Sven Behnke. “Evaluation of pooling operations in convolutional architectures for object recognition”. In: **International Conference on Artificial Neural Networks**. Springer. 2010, pp. 92–101.

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

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

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

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

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

[100] 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.

[101] 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.

[102] 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.

[103] 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.

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

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

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

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

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

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

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

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

[112] 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.

[113] Wojciech Zaremba. “An empirical exploration of recurrent network architectures”. In: (2015).

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

[115] Matthew D Zeiler et al. “Deconvolutional networks”. In: **Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on**. IEEE. 2010, pp. 2528–2535.

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 [44], Nagler [82], Quick [91], and Freedman [45]. 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.

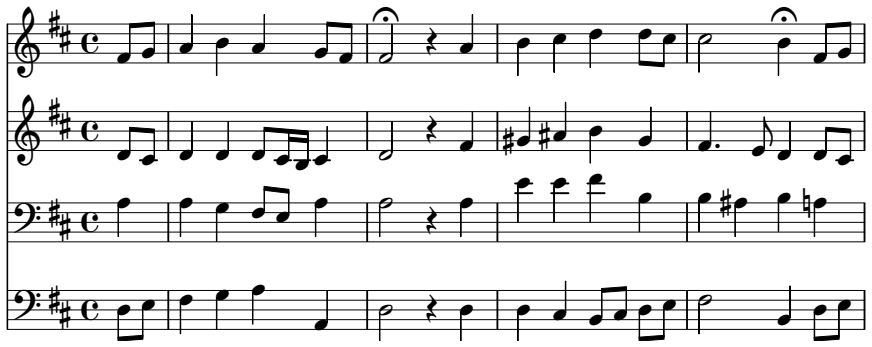


Fig. A.1 Sheet music representation of the first four bars of BWV 133.6

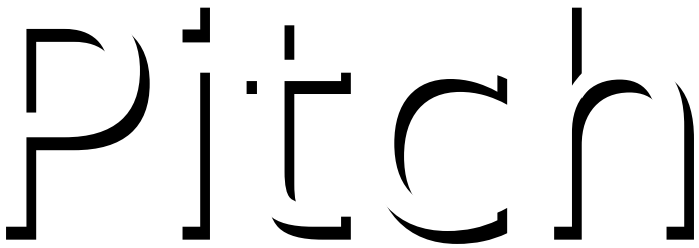


Fig. A.2 Terhardt's visual analogy for pitch. Similar to how the viewer of this figure may percieve contours not present, pitch describes subjective information received by the listener even when physical frequencies are absent.

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 [103], 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 is called an **interval** and an **octave** is an interval corresponding to the distance between

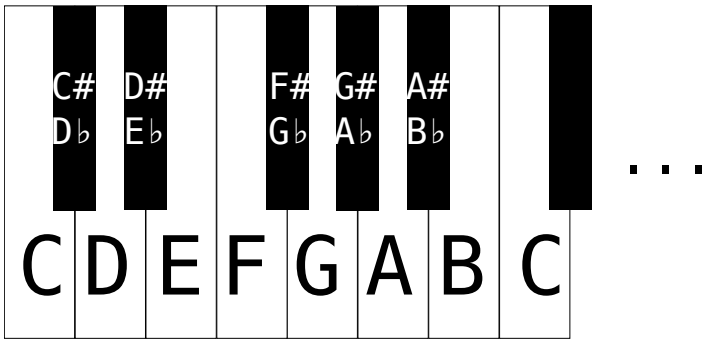


Fig. A.3 Illustration of an octave in the 12-note chromatic scale on a piano keyboard.

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 [31]. 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).

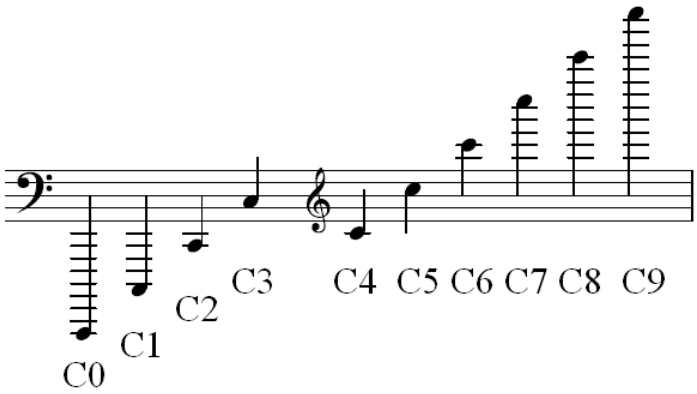


Fig. A.4 Scientific pitch notation and sheet music notation of *C* notes at ten different octaves.

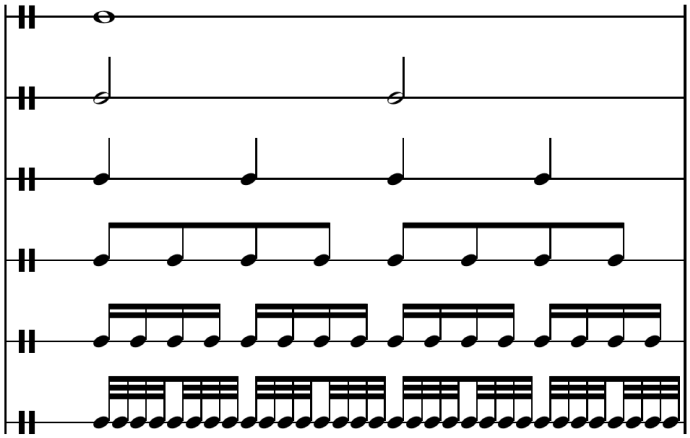


Fig. A.5 Comparison of various note durations [21]

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 **semibreve** (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 70, 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** [69]. Meter is implied in Western music, where bars establish perodic measures of equal length [56]. The meter of a score provides information about musical structure which can be used to predict chord changes and repepetition boundaries [22].

A.1.4 Piano roll notation

Figure A.6 shows the the same score from fig. A.1 on page 70 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" [64]. 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**.

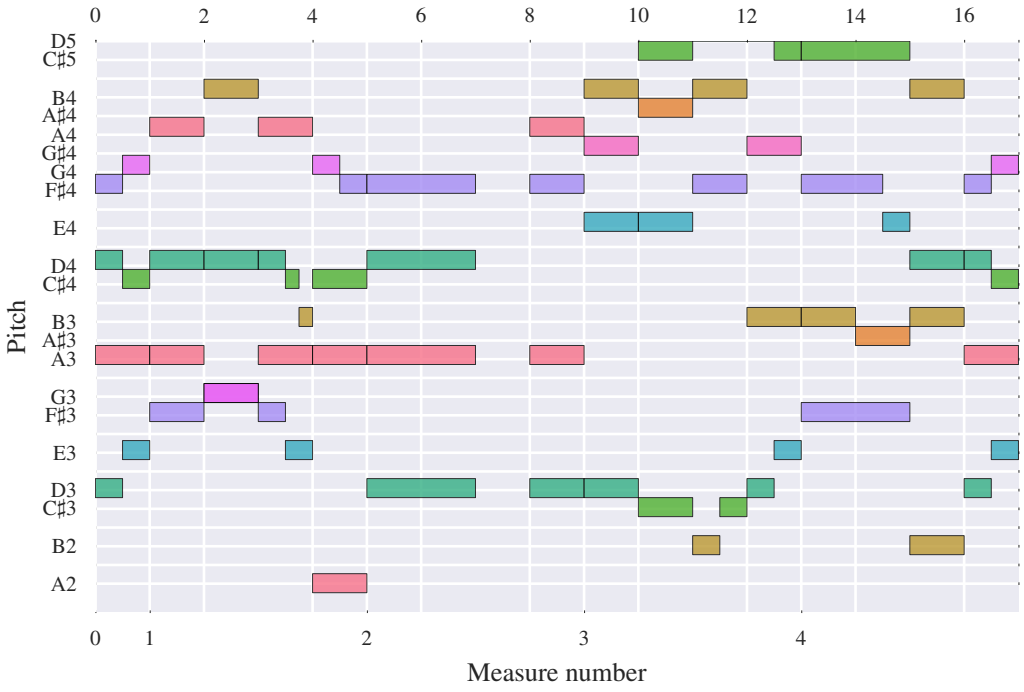


Fig. A.6 Piano roll notation of the music in fig. A.1

Table A.1 Pitch intervals for the two most important keys [45]. 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 [92]. In Western classical music, they typically consist of a **root note** whose pitch class forms a base from

Table A.2 Common chord qualities and their corresponding intervals [45]

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

which successive notes are built upon. The intervals between the pitch classes in a chord are 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 [66] or establishing a sense of conclusion at the end of phrases [92]. The most important cadences include:

Perfect cadence : V – I. The perfect cadence is described by Randel [92] 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 [66]

Interrupted cadepnce : V–vi. Also considered a weak cadence which invokes a "hanging" sensation prompting continuation [90].

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 5. 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.

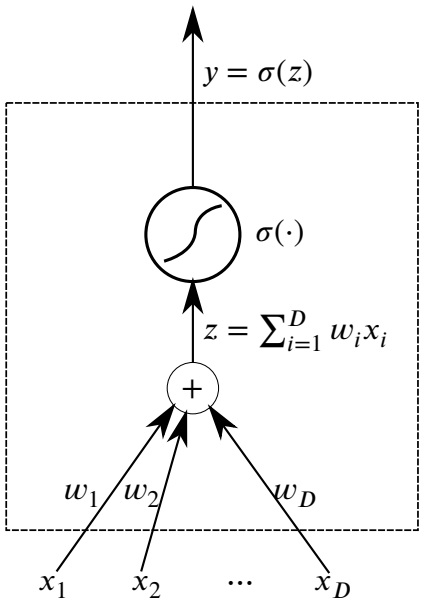


Fig. B.1 A single neuron first computes an activation z and then passes it through an activation function $\sigma(\cdot)$

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 [29], 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 [10].

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

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

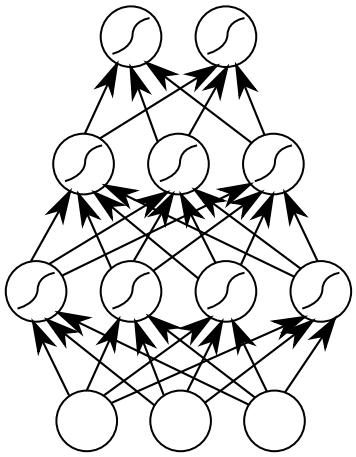


Fig. B.2 Graph depiction of a feedforward neural network with 2 hidden layers

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 [40]) or from the output layers to the hidden layers (Jordan networks [67]). Mathematically, a linear Elman-type RNN is a discrete time dynamical system commonly parameterized as:

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

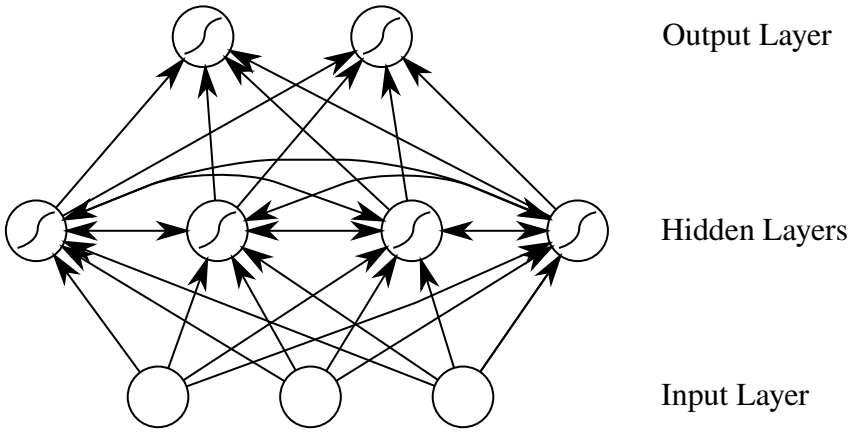


Fig. B.3 Graph representation of an Elman-type RNN.

(B.5)

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 7) replicated across time t.

C

Appendix C: Additional Proofs, Figures, and Tables

This section contains additional proofs, figures, and tables referenced from the body of this work. It is intended for readers who wish to examine our claims in greater detail.

C.1 Sufficient conditions for vanishing gradients

Following Pascanu, Mikolov, and Bengio [86], let $\|\cdot\|$ be any submultiplicative matrix norm (*e.g.* Frobenius, spectral, nuclear, Shatten p-norms). Without loss of generality, we will use the **operator norm** defined as

$$\|A\| = \sup_{x \in \mathbb{R}^n; x \neq 0} \frac{|Ax|}{|x|} \tag{C.1}$$

where $|\cdot|$ is the standard Euclidian norm.

Applying the definition of submultiplicativity to the factors of the product in eq. (2.4), we have that for any k

$$\left\| \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{h}_{k-1}} \right\| \leq \|\boldsymbol{W}_{hh}^\intercal\| \| \operatorname{diag}\left(\sigma'_{hh}(\boldsymbol{h}_{k-1})\right) \| \leq \gamma_{\boldsymbol{W}} \gamma_\sigma \tag{C.2}$$

where we have defined $\gamma_{\boldsymbol{W}} = \|\boldsymbol{W}_{hh}^{\mathsf{T}}\|$ and

$$\gamma_\sigma \coloneqq \sup_{h \in \mathbb{R}^n} \| \operatorname{diag}\left(\sigma'_{hh}(\boldsymbol{h})\right) \| \tag{C.3}$$

$$= \sup_{h \in \mathbb{R}^n} \max_i \sigma'_{hh}(\boldsymbol{h})_i \qquad \text{Operator norm of diag} \tag{C.4}$$

$$= \sup_{x \in \mathbb{R}} \sigma'_{hh}(x) \qquad \sigma_{hh} \text{ acts elementwise} \tag{C.5}$$

Substituting back into eq. (2.4), we find that

$$\left\| \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} \right\| = \left\| \prod_{t \geq i > k} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} \right\| \leq \prod_{t \geq i > k} \left\| \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} \right\| \leq (\gamma_{\boldsymbol{W}} \gamma_\sigma)^{t-k} \tag{C.6}$$

Hence, we see that a sufficient condition for vanishing gradients is for $\gamma_{\boldsymbol{W}} \gamma_\sigma < 1$, in which case $\left\| \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} \right\| \to 0$ exponentially for long timespans $t \gg k$.

If γ_σ is bounded, sufficient conditions for vanishing gradients to occur may be written as

$$\gamma_{\boldsymbol{W}} < \frac{1}{\gamma_\sigma} \tag{C.7}$$

This is true for commonly used activation functions (*e.g.* $\gamma_\sigma = 1$ for $\sigma_{hh} = \tanh$, $\gamma_\sigma = 0.25$ for $\sigma_{hh} = \text{sigmoid}$).

The converse of the proof implies that $\|\boldsymbol{W}_{hh}^{\mathsf{T}}\| \geq \frac{1}{\gamma_\sigma}$ are necessary conditions for $\gamma_{\boldsymbol{W}} \gamma_\sigma > 1$ and exploding gradients to occur.

C.1.1 Quantifying the effects of preprocessing

Related discussion is in section 4.1.1 on page 23.

Fig. C.1 Distribution of pitches used over Bach chorales corpus. Transposition has resulted in an overall broader range of pitches and increased the counts of pitches which are in key.

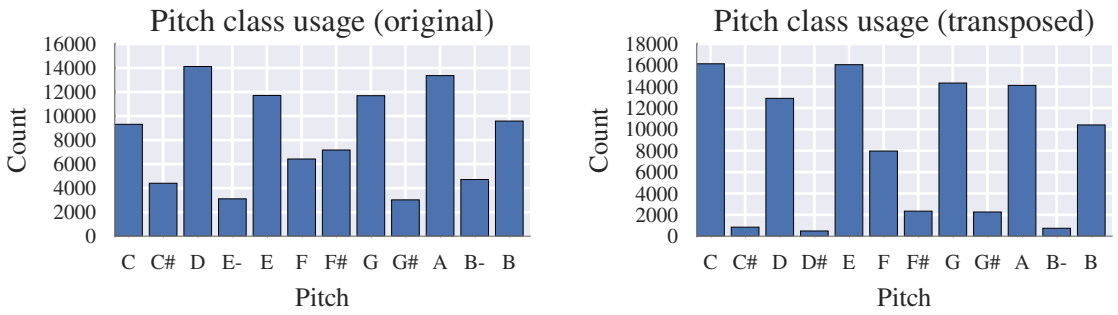


Fig. C.2 Distribution of pitch classes over Bach chorales corpus. Transposition has increased the counts for pitch classes within the C-major / A-minor scales.

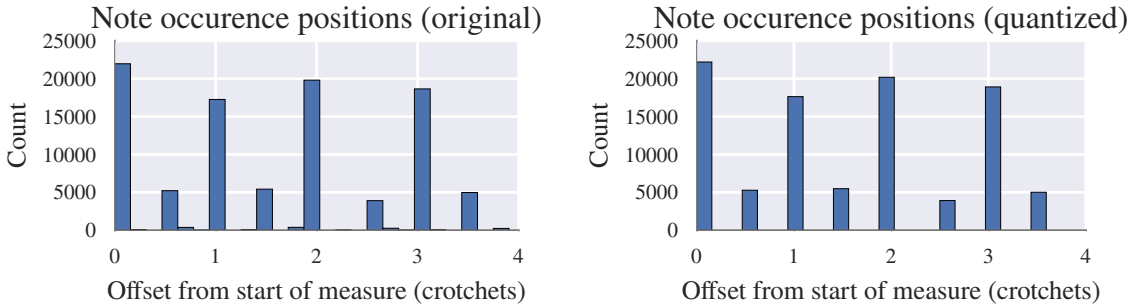


Fig. C.3 Meter is minimally affected by quantization due to the high resolution used for time quantization.

C.1.2 Discovering neurons specific to musical concepts

Related discussion is in section 5.1.3 on page 40.

C.1.3 Identifying and verifying local optimality of the overall best model

Related discussion is in section 4.2.5 on page 33.

Fig. C.5 Results of grid search (see Section 4.2.5) over LSTM sequence model hyperparameters

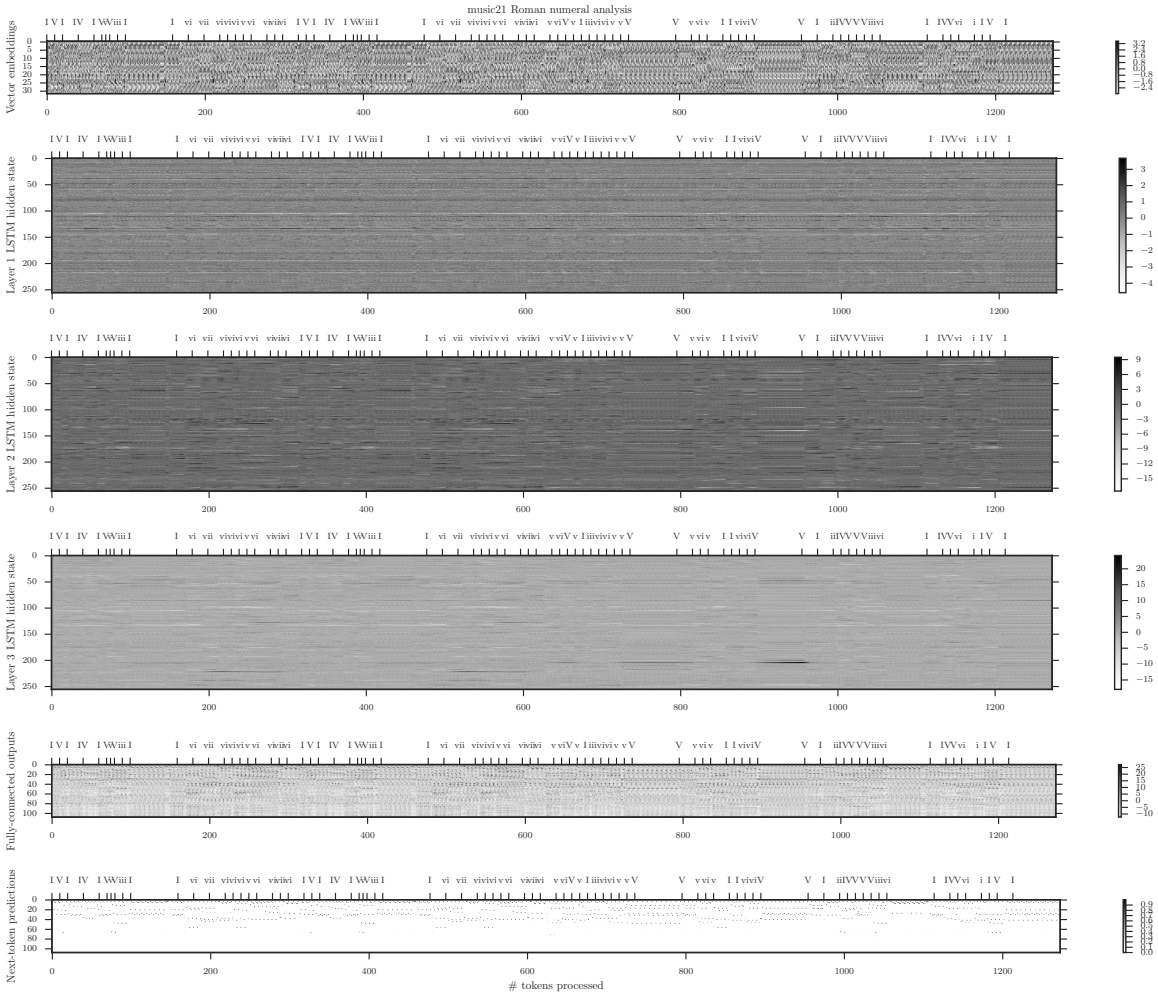


Fig. C.4 Neuron activations over time as the encoded stimulus is processed token-by-token

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
3.0	256.0	128.0	32.0	0.323781	0.477027
2.0	256.0	128.0	32.0	0.323668	0.479322
2.0	256.0	128.0	64.0	0.303158	0.482216
3.0	256.0	256.0	64.0	0.320361	0.484231
3.0	256.0	128.0	32.0	0.383811	0.484667
3.0	256.0	128.0	16.0	0.342955	0.484791
2.0	256.0	256.0	64.0	0.373641	0.485353
3.0	256.0	128.0	64.0	0.305290	0.486244
2.0	256.0	128.0	32.0	0.275125	0.486305
2.0	256.0	256.0	32.0	0.352257	0.486755
4.0	256.0	128.0	32.0	0.333133	0.487135
2.0	256.0	256.0	32.0	0.307188	0.487868
2.0	256.0	256.0	32.0	0.400955	0.489320
3.0	256.0	256.0	64.0	0.381868	0.489810
2.0	256.0	256.0	64.0	0.333356	0.491396
2.0	256.0	256.0	64.0	0.284248	0.491593
3.0	128.0	128.0	32.0	0.365171	0.492478
3.0	256.0	128.0	32.0	0.264723	0.492849
3.0	384.0	128.0	32.0	0.228556	0.495991
3.0	256.0	128.0	64.0	0.248987	0.496190
3.0	256.0	128.0	32.0	0.445840	0.498205
3.0	256.0	256.0	32.0	0.273567	0.499422
2.0	256.0	128.0	64.0	0.256022	0.500500
3.0	256.0	256.0	32.0	0.338776	0.501711
2.0	128.0	128.0	32.0	0.384075	0.501840
3.0	128.0	128.0	64.0	0.417780	0.501919
2.0	256.0	128.0	32.0	0.219939	0.502503
3.0	128.0	128.0	64.0	0.361381	0.503206
3.0	128.0	128.0	32.0	0.431771	0.503590
3.0	256.0	64.0	64.0	0.263001	0.503945
3.0	256.0	384.0	64.0	0.419091	0.504249
3.0	256.0	256.0	32.0	0.393463	0.506486
2.0	128.0	128.0	64.0	0.364640	0.506923

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
2.0	128.0	128.0	64.0	0.422178	0.507268
3.0	256.0	256.0	64.0	0.261563	0.507479
3.0	256.0	64.0	32.0	0.278916	0.507673
2.0	128.0	128.0	32.0	0.434552	0.508460
3.0	256.0	384.0	32.0	0.439684	0.514804
1.0	256.0	128.0	64.0	0.334873	0.517134
2.0	128.0	128.0	64.0	0.465061	0.520224
2.0	256.0	128.0	64.0	0.195905	0.521330
1.0	256.0	256.0	64.0	0.368281	0.522424
2.0	128.0	128.0	32.0	0.485346	0.522955
2.0	128.0	256.0	64.0	0.378280	0.525397
3.0	512.0	128.0	32.0	0.168366	0.525644
1.0	256.0	256.0	64.0	0.417803	0.525980
3.0	128.0	128.0	64.0	0.480340	0.526121
3.0	128.0	128.0	32.0	0.491876	0.527008
3.0	256.0	128.0	32.0	0.194120	0.528000
2.0	128.0	128.0	64.0	0.296537	0.528261
2.0	128.0	128.0	32.0	0.316390	0.529308
3.0	128.0	256.0	64.0	0.435649	0.529458
1.0	256.0	128.0	32.0	0.375717	0.529638
2.0	128.0	256.0	64.0	0.440450	0.529948
1.0	256.0	256.0	64.0	0.389651	0.531063
2.0	128.0	256.0	128.0	0.362561	0.533559
2.0	128.0	256.0	32.0	0.398919	0.533672
3.0	128.0	256.0	32.0	0.452009	0.536955
1.0	256.0	128.0	32.0	0.346140	0.538510
2.0	128.0	128.0	128.0	0.273516	0.539359
1.0	256.0	128.0	64.0	0.310597	0.539599
3.0	128.0	128.0	32.0	0.265842	0.539827
1.0	256.0	128.0	64.0	0.274568	0.541263
3.0	128.0	256.0	64.0	0.500697	0.544048
1.0	256.0	128.0	32.0	0.316189	0.545363
1.0	256.0	128.0	32.0	0.285714	0.546995

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
3.0	128.0	128.0	64.0	0.247192	0.549826
1.0	128.0	128.0	64.0	0.458142	0.550102
1.0	128.0	128.0	128.0	0.360038	0.550509
2.0	128.0	256.0	32.0	0.465110	0.550995
1.0	256.0	256.0	32.0	0.444180	0.551894
3.0	256.0	128.0	64.0	0.184959	0.552200
2.0	128.0	256.0	64.0	0.490587	0.552217
2.0	128.0	256.0	32.0	0.514900	0.553092
1.0	128.0	128.0	64.0	0.487574	0.553498
1.0	256.0	256.0	32.0	0.471938	0.553586
1.0	128.0	128.0	64.0	0.384282	0.554990
1.0	128.0	128.0	64.0	0.425469	0.555312
1.0	256.0	256.0	32.0	0.411686	0.555955
1.0	256.0	128.0	64.0	0.238860	0.556672
3.0	64.0	128.0	64.0	0.420250	0.559336
3.0	64.0	64.0	128.0	0.345705	0.559549
3.0	128.0	128.0	128.0	0.238071	0.562603
2.0	256.0	128.0	32.0	0.143647	0.563866
1.0	128.0	128.0	32.0	0.489160	0.564304
3.0	128.0	256.0	32.0	0.521478	0.566153
2.0	128.0	128.0	64.0	0.584950	0.567093
2.0	64.0	128.0	64.0	0.443393	0.567754
2.0	128.0	256.0	64.0	0.549169	0.568419
1.0	128.0	64.0	32.0	0.359041	0.569011
3.0	128.0	256.0	64.0	0.573862	0.570873
1.0	128.0	128.0	32.0	0.525982	0.571859
3.0	64.0	128.0	128.0	0.408074	0.572306
1.0	128.0	128.0	32.0	0.467434	0.572480
1.0	128.0	128.0	32.0	0.417764	0.573797
2.0	64.0	64.0	32.0	0.413944	0.573993
3.0	64.0	64.0	64.0	0.355615	0.574236
1.0	256.0	128.0	128.0	0.204964	0.574585
1.0	128.0	64.0	64.0	0.328927	0.575464

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
2.0	64.0	64.0	64.0	0.390597	0.575592
2.0	64.0	128.0	128.0	0.424735	0.575868
2.0	64.0	32.0	32.0	0.399389	0.577974
2.0	64.0	64.0	128.0	0.372478	0.578856
2.0	128.0	64.0	32.0	0.240288	0.580802
3.0	64.0	64.0	32.0	0.375478	0.582072
1.0	128.0	64.0	128.0	0.304245	0.582897
3.0	64.0	128.0	32.0	0.430421	0.582991
3.0	128.0	256.0	32.0	0.590133	0.585245
3.0	64.0	32.0	32.0	0.348150	0.585800
2.0	64.0	32.0	64.0	0.387047	0.589173
1.0	128.0	256.0	64.0	0.501138	0.593823
3.0	64.0	32.0	128.0	0.339394	0.594401
1.0	128.0	32.0	32.0	0.348193	0.595001
2.0	64.0	128.0	32.0	0.470837	0.597005
3.0	64.0	32.0	64.0	0.344404	0.597406
2.0	128.0	64.0	64.0	0.224014	0.597418
1.0	64.0	32.0	64.0	0.462827	0.597437
1.0	64.0	32.0	32.0	0.500014	0.598521
2.0	64.0	32.0	128.0	0.376624	0.600570
1.0	64.0	32.0	128.0	0.453646	0.604043
1.0	128.0	256.0	64.0	0.539087	0.604710
2.0	256.0	128.0	64.0	0.122328	0.606237
1.0	64.0	128.0	128.0	0.489255	0.607122
1.0	128.0	32.0	64.0	0.319029	0.609441
1.0	128.0	256.0	64.0	0.566182	0.610409
1.0	128.0	32.0	128.0	0.294204	0.613838
1.0	64.0	64.0	128.0	0.436633	0.615036
1.0	64.0	64.0	64.0	0.461935	0.616265
2.0	128.0	64.0	128.0	0.206896	0.620845
1.0	128.0	256.0	32.0	0.550056	0.627652
2.0	256.0	128.0	128.0	0.106181	0.631364
3.0	128.0	64.0	32.0	0.185779	0.633145

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
1.0	128.0	256.0	32.0	0.591930	0.638022
1.0	256.0	64.0	32.0	0.200897	0.640652
1.0	64.0	64.0	32.0	0.487779	0.643943
1.0	128.0	256.0	32.0	0.621720	0.647467
2.0	128.0	32.0	32.0	0.209044	0.647553
3.0	256.0	128.0	32.0	0.100153	0.650138
1.0	64.0	128.0	64.0	0.515733	0.653191
1.0	256.0	64.0	64.0	0.171567	0.657626
3.0	256.0	128.0	64.0	0.087426	0.660995
3.0	128.0	64.0	128.0	0.169560	0.663409
3.0	128.0	64.0	64.0	0.172871	0.670402
1.0	64.0	128.0	32.0	0.561724	0.670482
1.0	256.0	64.0	128.0	0.149129	0.672432
2.0	128.0	32.0	64.0	0.193615	0.688310
2.0	128.0	128.0	64.0	0.802259	0.696580
2.0	128.0	256.0	32.0	0.907374	0.701893
3.0	256.0	128.0	128.0	0.076598	0.711632
2.0	256.0	64.0	32.0	0.081134	0.716840
2.0	128.0	32.0	128.0	0.173684	0.727354
2.0	256.0	64.0	64.0	0.073675	0.742250
1.0	256.0	32.0	32.0	0.161496	0.743529
3.0	128.0	32.0	32.0	0.146775	0.752404
1.0	256.0	32.0	64.0	0.138145	0.755407
1.0	256.0	32.0	128.0	0.125931	0.757801
3.0	128.0	32.0	64.0	0.134530	0.770094
2.0	256.0	64.0	128.0	0.063084	0.797383
3.0	128.0	32.0	128.0	0.129410	0.801131
3.0	256.0	64.0	64.0	0.048852	0.823713
3.0	256.0	64.0	32.0	0.052363	0.848516
2.0	256.0	32.0	32.0	0.058634	0.874037
3.0	256.0	64.0	128.0	0.044448	0.876398
2.0	256.0	32.0	128.0	0.049791	0.888397
2.0	256.0	32.0	64.0	0.050012	0.898488

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
3.0	256.0	32.0	32.0	0.037417	0.960396
3.0	256.0	32.0	64.0	0.034403	0.988554
3.0	256.0	32.0	128.0	0.036275	0.990457

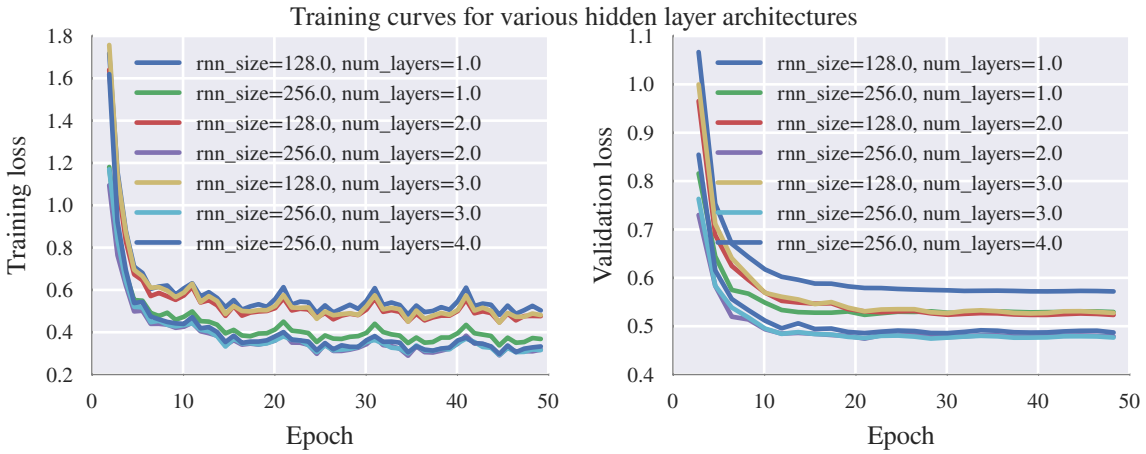


Fig. C.6 `rnn_size=256` and `num_layers=3` yields lowest validation loss.

Fig. C.7 Validation loss improves initially with increasing network depth but deteriorates after > 3 layers.

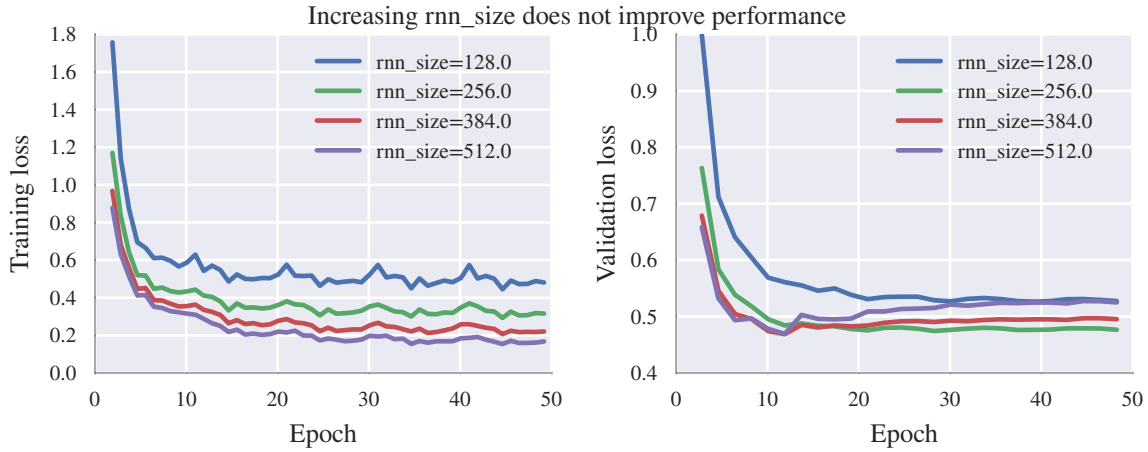


Fig. C.8 Validation loss improves initially with higher-dimensional hidden states but deteriorates after > 256 dimensions.

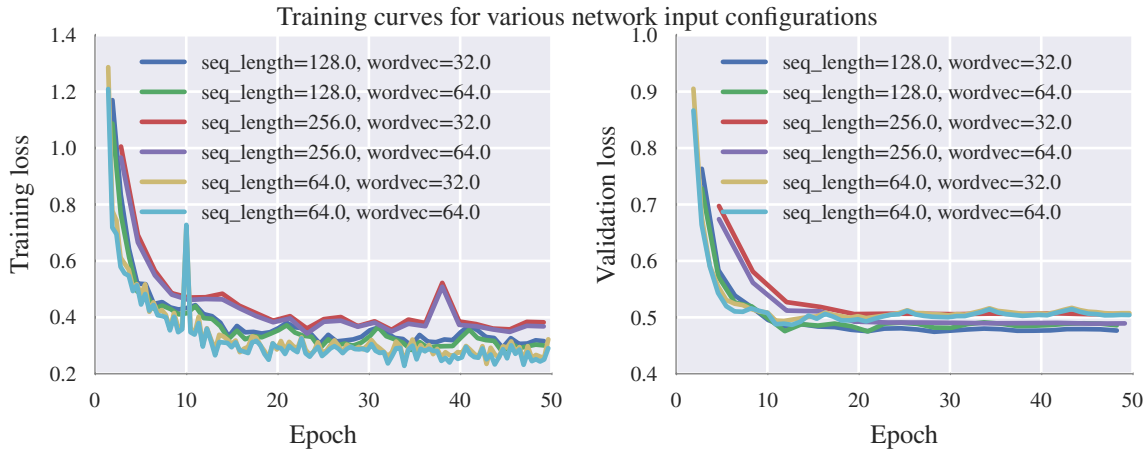


Fig. C.9 `seq_length=128` and `wordvec=32` yields lowest validation loss.

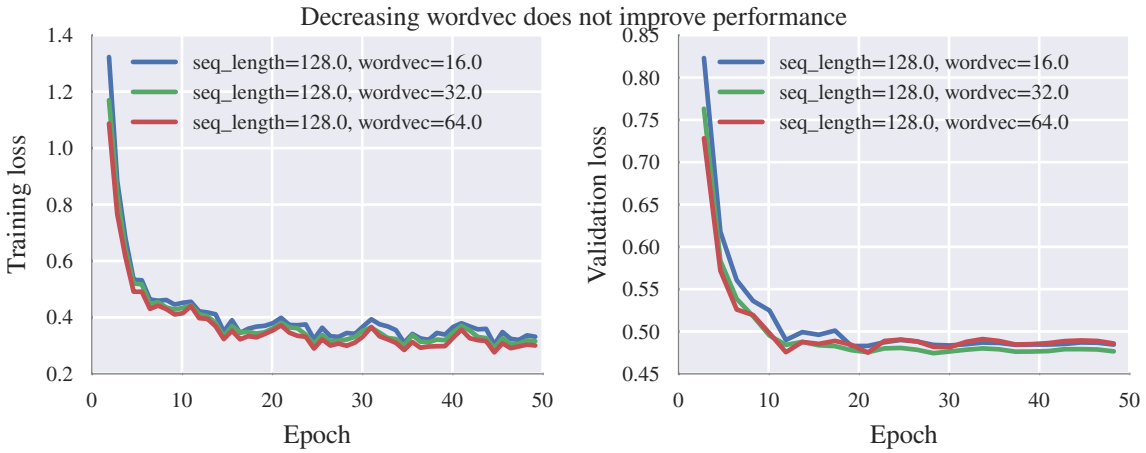
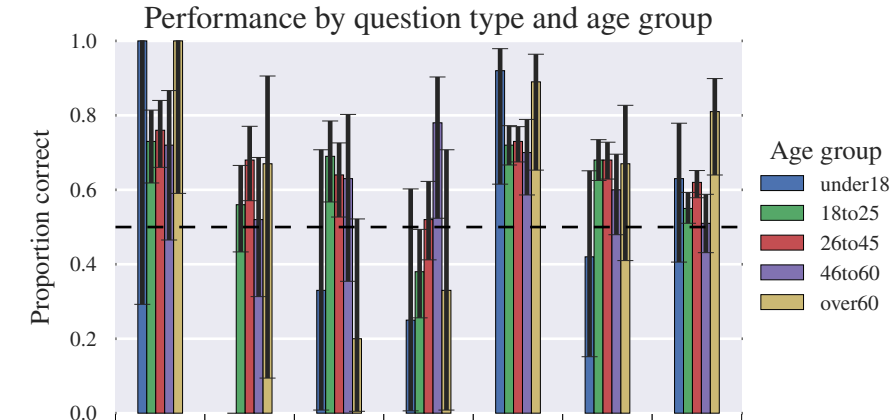


Fig. C.10 Perturbations about `wordvec=32` do not yield significant improvements.

C.1.4 Additional large-scale subjective evaluation results

Related discussion is in section 7.2 on page 51.



	S	A	T	B	AT	ATB	SATB
under18	1.0	0.0	0.33	0.25	0.92	0.42	0.63
18to25	0.73	0.56	0.69	0.38	0.72	0.68	0.55
26to45	0.76	0.68	0.64	0.52	0.73	0.68	0.62
46to60	0.72	0.52	0.63	0.78	0.7	0.6	0.51
over60	1.0	0.67	0.2	0.33	0.89	0.67	0.81

Fig. C.11 Proportion of correct responses for each question type and age group.