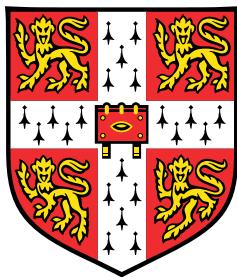


Deep generative modelling of Bach chorales



Feynman Liang

Supervisor: **Prof. Bill Byrne**

Advisors: Dr. Matthew Johnson
Dr. Jamie Shotton

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Masters of Philosophy

Churchill College

August 2016

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

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

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Total word count:

fliang: Fill this in at the end

Signed:

Date:

Feynman Liang
August 2016

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

Acknowledgements

And I would like to acknowledge ...

Mark Gotham : for providing his musical expertise and time to answer my questions direct the project.

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

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

Abstract

This is where you write your abstract ...

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

Table of contents

List of figures	15
List of tables	19
Nomenclature	21
1 Introduction	23
2 Background	27
2.1 A primer on Western music theory	28
2.1.1 Notes: the basic building blocks	29
2.1.2 Key signature	32
2.1.3 Polyphony, chords, and chord progressions	32
2.2 Neural sequence probability modeling	34
2.2.1 Neurons: the basic computation unit	34
2.2.2 Feedforward neural networks	35
2.2.3 Recurrent neural networks	36
2.2.4 Modeling assumptions	40
2.2.5 Training RNNs: backpropogation through time	41
2.2.6 Long short term memory: solving the vanishing gradient	44
3 Related Work	47
3.1 Machine learning on musical data	47
3.2 Models for music synthesis	48
3.2.1 Symbolic rule-based methods	48
3.2.2 Connectionist methods	49
3.2.3 Hybrid methods	49
3.2.4 LSTM music synthesis models	50

3.3 Generative modelling of Bach Chorales	51
4 Automatic composition	53
4.1 Constructing a corpus of encoded scores	53
4.1.1 Preprocessing	54
4.1.2 Sequential encoding of musical data	56
4.2 Design and validation of a generative model for music	57
4.2.1 Training and evaluation criteria	57
4.2.2 Comparing memory cells for music data	58
4.2.3 Optimizing the LSTM architecture	59
4.3 Results	60
4.4 Accelerating model training with GPUs	61
4.5 Other applications	61
5 Chorale harmonization	69
5.1 Background	69
5.2 Harmonizing	70
5.3 Datasets	72
5.4 Results	72
5.4.1 “Bachifying” other music	73
6 Large-scale subjective evaluation	75
6.1 Evaluation framework design	75
6.1.1 Software architecture	75
6.1.2 User interface	76
6.1.3 Question generation	77
6.2 Results	78
6.2.1 Participant backgrounds and demographics	78
6.2.2 BachBot’s performance results	78
6.3 User feedback	82
6.4 Competitive analysis of large-scale evaluation methodologies	83
7 Analysis of musical concepts learned by the model	85
7.1 Investigation of neuron activation responses to applied stimulus	85
7.1.1 Pooling over frames	88
7.1.2 Probabilistic piano roll: likely variations of the stimulus	88
7.1.3 Neurons specific to musical concepts	91

Table of contents	13
8 Summary and Conclusions	95
8.1 Contributions	95
8.2 Findings and conclusions	95
8.3 Future work	95
References	97
Appendix A Installing the CUED class file	111
9 Graveyard	113
9.1 Neural Networks	113
9.2 RNNs	118
9.3 Sequence probability modelling	119
9.4 Related work	120
9.5 LSTMs: background and motivation	120
9.5.1 Representation of music data	120
9.5.2 Evaluation of models	121
9.6 Automatic Composition	121
9.6.1 Multi-GPU implementation	121
9.7 Token-level embeddings	122
9.7.1 Variable-length embeddings	122

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

List of figures

2.1	Sheet music representation of some bars from a musical score (BWV133.6) with articulation instructions removed.	29
2.2	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.	29
2.3	Illustration of an octave in the 12-note chromatic scale on a piano keyboard. .	30
2.4	Scientific pitch notation and sheet music notation of C notes at ten different octaves.	31
2.5	Comparison of various note durations.	31
2.6	A single neuron first computes an activation z and then passes it through an activation function $\sigma(\cdot)$	34
2.7	Graph depiction of a feedforward neural network with 2 hidden layers	35
2.8	Graph representation of an Elman-type RNN.	36
2.9	Equivalent formulation of an Elman-type RNN treating the time-delayed hidden state \mathbf{h}_{t-1} as additional inputs to a feedforward network	37
2.10	Signal flow diagram representation of a single-layer RNN and its corresponding directed acyclic graph after unrolling	38
2.11	Template and unrolling of a stacked 2-layer RNN	39
2.12	The gradients passed along network edges during BPTT.	42
2.13	Schematic for a single LSTM memory cell. Notice how the gates i_t , o_t , and f_t control access to the constant error carousel (CEC).	45
4.1	First 4 bars of JCB Chorale BWV 133.6 before (top) and after (bottom) pre-processing. 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. . . .	55

4.2	Piano roll representation of the same 4 bars from ?? before and after pre-processing. Notice the transposition of key as well as quantization of the C5 and D5 notes (after transposition) in bar 3 of the Soprano part.	63
4.3	Pitches before and after key standardization	64
4.4	Pitch classes before and after key standardization	64
4.5	Effects of time quantization on note durations	64
4.6	Effects of time quantization on meter	65
4.7	theanets-architecture	65
4.8	torch-rnn-best-model-Trace	65
4.9	torch-rnn-Dropout	66
4.10	torch-rnn-network-params	66
4.11	torch-rnn-network-params-num-Layers	67
4.12	torch-rnn-network-params-rnn-size	67
4.13	torch-rnn-input-params	68
4.14	torch-rnn-input-params-Wordvec	68
5.1	Error rates for harmonization tasks	72
5.2	Happy birthday soprano melody, ATB harmonized by BachBot	73
5.3	Twinkle-twinkle soprano melody, ATB harmonized by BachBot	74
6.1	The first page seen by a visitor of http://bachbot.com	76
6.2	User information form presented after clicking “Test Yourself”	77
6.3	Question response interface used for all questions	78
6.4	Geographic distribution of participants	79
6.5	Demographics of participants	80
6.6	responses-Mask	80
6.7	responses-mask-MusicExperience	81
6.8	responses-mask-Agegroup	82
6.9	Proportion of correct responses broken down by individual questions.	83
7.1	<i>Top:</i> The preprocessed score (BWV 133.6) used as input stimulus with Roman numeral analysis annotations obtained from music21; <i>Bottom:</i> The same stimulus represented on a piano roll	86
7.2	Neuron activations over time as the encoded stimulus is processed token-by-token	87
7.3	Neuron activations over time pooled over frames	89
7.4	Top: piano roll of stimulus (included for reference); Bottom: probabilistic piano roll	90

7.5 Activation profiles of neurons within our model which have learned high-level musical concepts	92
1 Results of grid search (see ??) over LSTM sequence model hyperparameters	104
9.1 Single feedfoward neural network layer	113
9.2 2-layer feedforward neural network	114
9.3 Single LSTM unit	119
9.4 PCA embedding of note tokens	122
9.5 tSNE embedding of note tokens	123

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

List of tables

2.1	Pitch intervals for the two most important modes[38]. The pitches in a scale can be found by starting at the tonic and successively offsetting by the given pitch intervals.	32
2.2	Common chord qualities and their corresponding intervals[38]	33
4.1	Perplexities of baseline N -gram language models	62
6.1	Composition of questions on http://bachbot.com	77

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

Nomenclature

Acronyms / Abbreviations

A Alto

AT Alto and Tenor

ATB Alto, Tenor, and Bass

B Bass

FER Frame Error Rate

GPU Graphics Processing Unit

LBM Lattice Boltzmann Method

RNN Recurrent Neural Network

SATB Soprano, Alto, Tenor, and Bass

S Soprano

TER Token Error Rate

T Tenor

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

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 [11]

1

1

Introduction

2

Bringing together ideas from language modelling, deep learning, and music psychology, we develop a generative model for music and conduct a large-scale subjective evaluation. Our results validate our success: participants were only 5% more likely to identify an original Bach composition from a sample from BachBot. To our knowledge, no prior work in automatic composition has carried out a study at this scale.

3

4

5

6

7

Our fundamental question is this: have advances in deep learning enabled construction of musical models capable of deceiving human listeners. To answer this question, we build a model incorporating the current state-of-the-art in deep neural sequence modelling and conduct a large-scale musical Turing test. Our results suggest an undeniable yes.

8

9

10

11

Our contributions include:

12

1. A note-by-note sequential representation for polyphonic music amenable to processing with standard sequence models

13

14

2. A rigorous investigation of how recent deep learning advances such as dropout [88], batch normalization [57], and new RNN architectures can be applied to improve probabilistic modelling of music data

15

16

17

1 3. A connectionist model for Bach chorales which avoids domain-specific feature engi-
2 neering and is capable of composing, completing, harmonizing, and scoring polyphonic
3 scores

4 4. The first large-scale music Turing test with over

5 fliang: XXX

6 participants

7 While deep learning has revolutionized computer vision and natural language processing,
8 its applications to other domains are still emerging. This dissertation is concerned with the
9 applications of deep learning to a new problem domain: music scores.

10 In this work, we investigate how sequence probability models parameterized by deep re-
11 current neural networks can be used as generative models over scores of music. Such a model
12 has a variety of applications within computational music theory. The aim of this work is to
13 investigate applications on two particular tasks: melody harmonization and automatic compo-
14 sition.

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

21 A more difficult task is automatic composition, where the composer is tasked with produc-
22 ing an original composition of a particular musical style. The open nature of this task enables
23 a composer to simultaneously demonstrate their creativity along with understanding of music
24 theory. However, this lack of constraints and loose definition of musical style makes it more
25 difficult to evaluate the quality of the output. To apply a generative model towards this task,
26 we can train the model to assign larger probability mass to stylistically similar scores and then
27 sample the model to generate a novel composition.

28 While our modeling framework is capable of modeling any MIDI-encodeable music score,
29 we focus our study on chorales by Johann Sebastian Bach. These provide a relatively large cor-
30 pus by a single composer, are well understood by music theorists, and are routinely used when
31 teaching music theory. The aim is to build an automatic music composition system capable of
32 imitating Bach's compositional style on both harmonization and automatic composition tasks.

33 We will examine how design decisions made when constructing probability models over
34 music affect the musical characteristics of generated samples, investigate practical matters en-

countered with parallel training and sampling across multiple GPUs, and benchmark how well
our final system performs on human test subjects.

With advances in computing and progress in modeling methods and algorithms, computational
modeling has started to provide novel insights into various musical phenomena. By offering a method for quantitatively testing theories, it can help us to learn more about the
various cognitive and perceptual processes related to music comprehension, production, and
style.

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

2

Background

fliang: Similarity to language modeling, char-rnn vs word-rnn similar so note-rnn vs chord-rnn should be as well. Benefit is that chord vocabulary MUCH larger than natural language

fliang: Try mimicking Franklin [37]

fliang: Motivation: 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 Cooper and Meyer [16].

fliang: Describe automatic composition vs harmonization vs analysis/scoring

Generative models are preferable because they can be:

1. Conditioned for harmonization
2. Sampled for automatic composition
3. Queried for analysis/scoring

Automatic composition is described by Papadopoulos and Wiggins [76] as:

a sequence (set) of rules (instructions, operations) for solving (accomplishing) a [particular] problem (task) [in a finite number of steps] of combining musical parts (things, elements) into a whole (composition)

¹ Pearce et al. [78] emphasizes *algorithmic composition* should “be applied exclusively to activities in which the motivations have this artistic character.”

³ fliang: Describe piano roll vs sheet music

⁴ *Piano roll* music transcriptions (??) are quantized both in time ($t \in T$) and note frequencies ($n \in N$). frequencies quantized to a piano roll.

⁶ fliang: Motivate quantization with Western music

⁷ .

⁸ We can represent a piano roll transcription as a high-dimensional vector $X_{t,n} \in \mathbb{R}^{|T| \times |N|}$ where $X_{t,n}$ denotes the note velocities for note n at time t .

¹⁰ fliang: Describe MIDIs 127 quantized pitches, isomorphic to musicXML

¹¹ fliang: Describe Bach and chorales, 4 parts, soprano lead with ATB harmony, regular phrases, fermatas to denote phrase ends. Background section in chorale harmonization.

¹² fliang: Define meter

¹³ Eck and Lapalme [28] first addressed. *Meter* is the sense of a periodic pattern of strong and weak beats which arise from periodic articulation of notes in common locations. It is implied in Western music, where bars establish periodic measures of equal length [49]. Meter provides us with key information about musical structure which can be used to predict chord changes and repetition boundaries [16].

¹⁸ This chapter provides background information on two topics heavily utilized throughout this dissertation: Western music theory and recurrent neural networks. It introduce the definitions and notation used in later sections.

²¹ 2.1 A primer on Western music theory

²² 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. ??) 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.

²⁷ 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.

2.1 A primer on Western music theory

29



Fig. 2.1 Sheet music representation of some bars from a musical score (BWV133.6) with articulation instructions removed.

2.1.1 Notes: the basic building blocks

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

Pitch

Though pitch is closely related to physical frequency of vibration of a waveform (as measured in Hertz), pitch its a perceptual property whose semantic meaning is derived from a listener's perception. This distinction has been scrutinized by Terhardt [95], whose visual analogy in ?? 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.

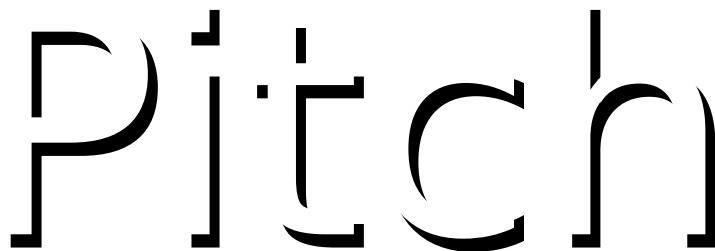


Fig. 2.2 Terhardt's visual analogy for pitch. Similar to how the viewer of this figure may percieve contours not present, pitch describes subjective information received by the listener even when physical frequencies are absent.

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 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 perceived 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) presently the most widely used method [25].

fliang: Talk about well-tempered tuning and bach

10

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. ?? illustrates how these pitch classes map to keys on a piano.

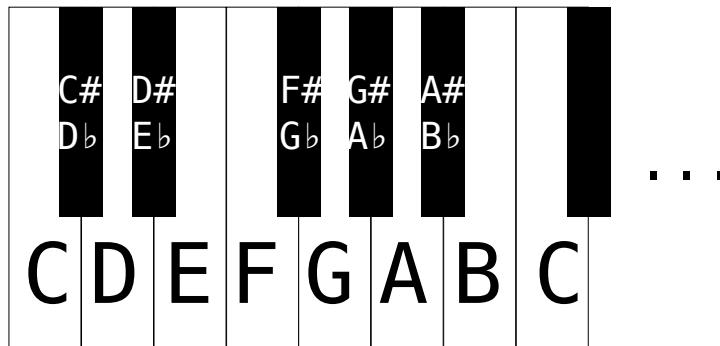


Fig. 2.3 Illustration of an octave in the 12-note chromatic scale on a piano keyboard.

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 ??). 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).

fliang: Cite wiki scientific pitch notation

26

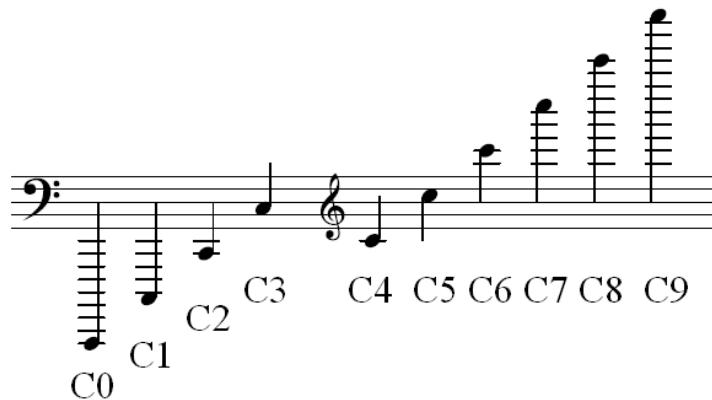


Fig. 2.4 Scientific pitch notation and sheet music notation of *C* notes at ten different octaves.

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 ?? . 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.

fliang: Cite wiki Whole note



Fig. 2.5 Comparison of various note durations.

¹ 2.1.2 Key signature

² *Tonal music* is a genre of music encompassing most of Western classical music. It is characterized by the prevalence of one pitch class (the *tonic*) around which the remainder of the piece ³ is built.

⁵ A basic concept within tonal music is *mode*, which defines a subset of pitch classes that ⁶ are “in key” with respect to the tonic. ?? illustrates the pitch intervals between adjacent pitch ⁷ classes within two important modes: the *major* (Ionian, I) and *minor* (Aeolian, VI) modes.

⁸ The choice of tonic and mode is collectively referred to as the *key signature*.

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

Table 2.1 Pitch intervals for the two most important modes[38]. The pitches in a scale can be found by starting at the tonic and successively offsetting by the given pitch intervals.

⁹ 2.1.3 Polyphony, chords, and chord progressions

¹⁰ Whereas *monophonic* music is characterized by the presence of a single *part* sounding at most ¹¹ one note at any given time, *polyphonic* music contains multiple parts potentially sounding ¹² multiple pitches at the same time. Just as notes form the basis of monophonic music, chords ¹³ are the fundamental building blocks for polyphonic music.

¹⁴ Chords: basic units for representing simultaneously sounding notes

¹⁵ A *chord* is a collection of three or more pitches all sounding simultaneously[82]. In Western ¹⁶ classical music, they typically consist of a *root note* whose pitch class forms a base from which ¹⁷ 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 ¹⁸ same chord (e.g. octave choices for each pitch class) are called *voicings*. ¹⁹

²⁰ ?? lists some common chord qualities and their corresponding intervals from the root note. ²¹ Chord names are given as a root pitch class followed by a quality, for example: *C* major, *A* ²² minor, or *G* half-diminished 7th.

²³ The lowest note in a chord is called the *bass note* and is oftentimes the root note. However, ²⁴ alternative voicings called *inversions* can place the root note on a different octave and cause ²⁵ the bass and root notes to differ.

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

Table 2.2 Common chord qualities and their corresponding intervals[38]

Chord progressions and phrases

Sequences of chords are called *chord progressions*, which are oftentimes grouped with adjacent progressions within a score into coherent units called *phrases*. Many psychoacoustic phenomena such as stability, mood, and expectation can be attributed to phrase structure and choice of chord progressions. For example, chord progressions called *harmonic cadences* are commonly used to conclude a phrase and create a sense of resolution[82]. Another important example are *modulations*, where a chord progression is used to transition the music into a different key signature.

As chords can be overlapping and contain notes straddling between two chords or involve uncommon chord qualities, identifying chord progressions involves a degree of subjectivity. A common method for analyzing chord progressions is *roman numeral analysis*, where I denotes the tonic pitch class, successive roman numerals correspond to successive pitch classes in the key, and capitalization is used to denote major/minor. For example, the chord progression *C* major – *A* minor – *D* major 7th – *G* major in the *C* major key signature would be represented as *I* – *ii* – *II maj7* – *V*.

Transposition invariance

A common theme throughout our discussion has been ambiguity of the tonic. When discussing key signatures, the mode was defined using intervals relative to a choice of tonic. Similarly, roman numeral analysis of chord progressions is also conducted relative to a tonic. Even if a chord progression and tonic are both transposed by arbitrary pitch interval, the roman numeral analysis will remain unchanged.

¹ The *transposition invariance* of chord progressions and modes is an important property of
² music. It enables us to offset an entire score by an arbitrary pitch interval without affecting the
³ important psychoacoustic qualities captured by chord progressions and choice of mode.

⁴ 2.2 Neural sequence probability modeling

⁵ Our work in later sections make heavy use of neural networks. In this section, we briefly review
⁶ the relevant concepts and set up notation.

⁷ 2.2.1 Neurons: the basic computation unit

⁸ Neurons are the basic abstraction which are combined together to form neural networks. A
⁹ *neuron* is a parametric model of a function $f : \mathbb{R}^D \rightarrow \mathbb{R}$ from its D -dimensional input \mathbf{x} to its
¹⁰ output y . Our neurons will be defined as

$$\text{¹¹ } f(\mathbf{x}) := \sigma(\langle \mathbf{w}, \mathbf{x} \rangle) \quad (2.1)$$

¹² which can be viewed as an inner product with *weights* \mathbf{w} to produce an *activation* $z := \langle \mathbf{w}, \mathbf{x} \rangle \in$
¹³ \mathbb{R} which is then squashed to a bounded domain by a non-linear **activation function** $\sigma : \mathbb{R} \rightarrow$
¹⁴ $[L, U]$. This is visually depicted in ?? , which also makes apparent the interpretation of weight
¹⁵ w_i as the sensitivity of the output y to the input x_i .

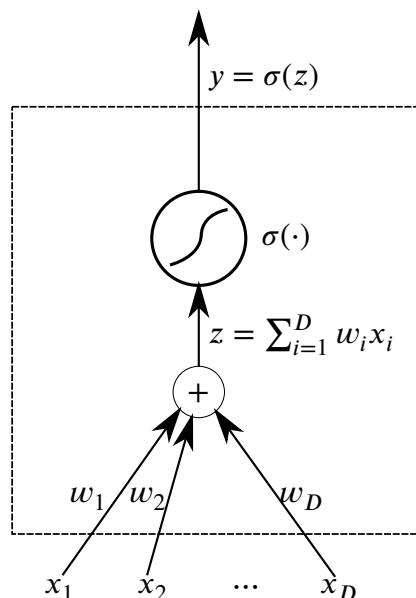


Fig. 2.6 A single neuron first computes an activation z and then passes it through an activation function $\sigma(\cdot)$

2.2.2 Feedforward neural networks

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

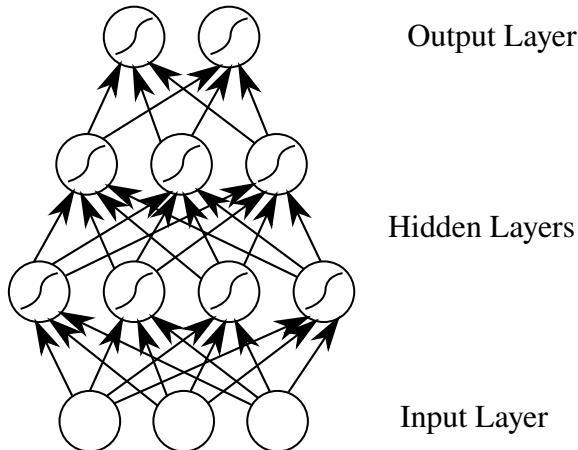


Fig. 2.7 Graph depiction of a feedforward neural network with 2 hidden layers

Although a single hidden layer is theoretically sufficient for a universal function approximator[22], the number of hidden units to guarantee reported theoretical bounds are usually unfeasibly 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[9].

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 $\mathbf{W}_{l,l+1}$ denote the weights between layers l and $l + 1$. The original neural network computes the function

$$\sigma(\mathbf{W}_{L,L-1}\sigma(\mathbf{W}_{L-1,L-2}\cdots\sigma(\mathbf{W}_{2,1}\mathbf{x})\cdots)) \quad (2.2)$$

After removing the activation functions σ , we are left with

$$\mathbf{W}_{L,L-1}\mathbf{W}_{L-1,L-2}\cdots\mathbf{W}_{2,1}\mathbf{x} = \tilde{\mathbf{W}}\mathbf{x} \quad (2.3)$$

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

¹ 2.2.3 Recurrent neural networks

² While feedforward neural networks provide a flexible model for approximating arbitrary functions,
³ they require a fixed-dimension input \mathbf{x} and hence cannot be directly applied to sequential
⁴ data $\mathbf{x} = (\mathbf{x}_t)_{t=1}^T$ where T may vary.

⁵ A naive method for extending feedforward networks would be to independently apply a
⁶ feedforward network to compute $\mathbf{y}_t = f(\mathbf{x}_t, \theta)$ at each timestep $1 \leq t \leq T$. However, this
⁷ approach is only correct when each output \mathbf{y}_t depends only on the input at the current time \mathbf{x}_t ,
⁸ and is independent of all prior inputs $\{\mathbf{x}_k\}_{k < t}$. This assumption is false in musical data: the
⁹ current musical note usually is highly dependent on the sequence of notes leading up to it.

¹⁰ This shortcoming motivates *recurrent neural networks* (RNNs), which generalize feed-
¹¹ forward networks by introducing time-delayed recurrent connections between hidden layers
¹² (Elman networks [31]) or from the output layers to the hidden layers (Jordan networks [58]).
¹³ Mathematically, an (Elman-type) RNN is a discrete time dynamical system commonly param-
¹⁴ eterized as:

$$\mathbf{h}_t = \mathbf{W}_{xh}\sigma_{xh}(\mathbf{x}_t) + \mathbf{W}_{hh}\sigma_{hh}(\mathbf{h}_{t-1}) \quad (2.4)$$

$$\mathbf{y}_t = \mathbf{W}_{hy}\sigma_{hy}(\mathbf{h}_t) \quad (2.5)$$

¹⁷ where $\sigma_{..}(\cdot)$ are activation functions acting element-wise and $\theta = \{\mathbf{W}_{xh}, \mathbf{W}_{hh}, \mathbf{W}_{hy}\}$ are the
¹⁸ learned parameters. ?? provides a graphical illustration of such a network. Notice that apart
¹⁹ from the edges between hidden nodes, the network is identical to a regular feedforward network
²⁰ (??).

fliang: Why do we use Elman

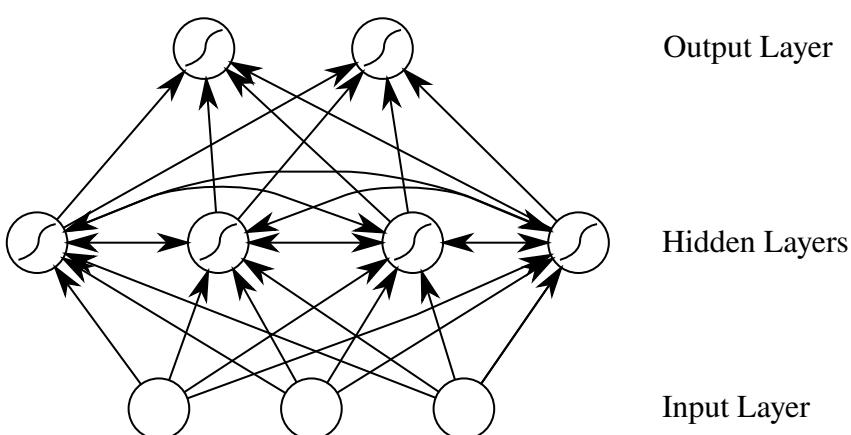


Fig. 2.8 Graph representation of an Elman-type RNN.

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

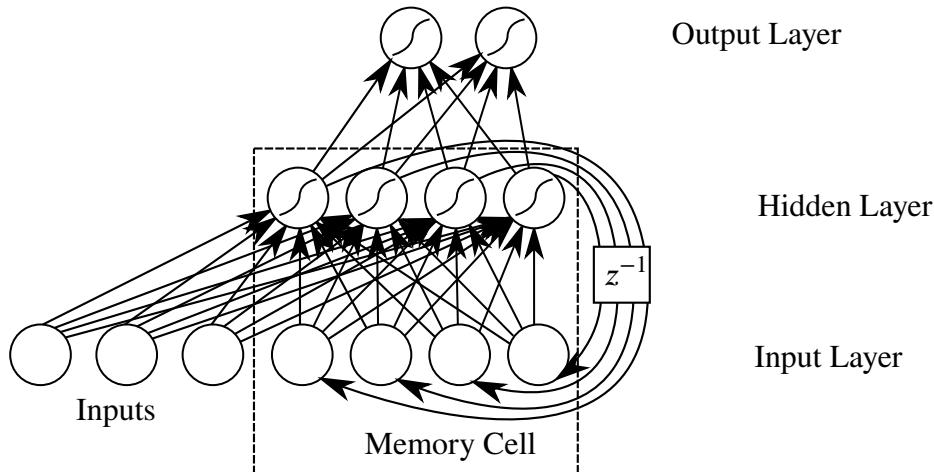


Fig. 2.9 Equivalent formulation of an Elman-type RNN treating the time-delayed hidden state \mathbf{h}_{t-1} as additional inputs to a feedforward network

Memory cell abstraction

Notice that ?? delineates the recurrent hidden state from the rest of the diagram, introducing an abstraction called a *memory cell*. This allows us to abstract away how \mathbf{y}_t and \mathbf{h}_t are computed from \mathbf{x}_t and \mathbf{h}_{t-1} , enabling discussion of RNNs applicable to many different implementations. Concretely, A memory cell is an interface which for each timestep t :

Inputs

- The current element in the input sequence \mathbf{x}_t
- The previous hidden state \mathbf{h}_{t-1}

Outputs

- The next hidden state $\mathbf{h}_t = f_h(\mathbf{x}_t, \mathbf{h}_{t-1})$
- The outputs $\mathbf{y}_t = f_y(\mathbf{h}_t)$.

In future diagrams, we will abstractly represent the memory cell as a node labelled with \mathbf{h} .

¹ **Unrolling into a DAG**

² One important operation on RNNs is *unrolling* the template in ?? into a chain of T replications
³ with connected hidden states (??). This removes the time-delayed recurrent, converting the
⁴ RNN into a finite directed acyclic graph where nodes represent pieces of data and edges $s \rightarrow t$
⁵ indicate that t is a function of s . This is identical to a feedforward network, justifying the view
⁶ of RNNs as a dynamically-sized feedforward network with T layers and parameters tied across
⁷ all layers.

fliang: Talk about how view of unrolled RNN as DNN inspires cross-polination e.g. gating to let information flow deeper: LSTM gates and highway networks, grid LSTMS

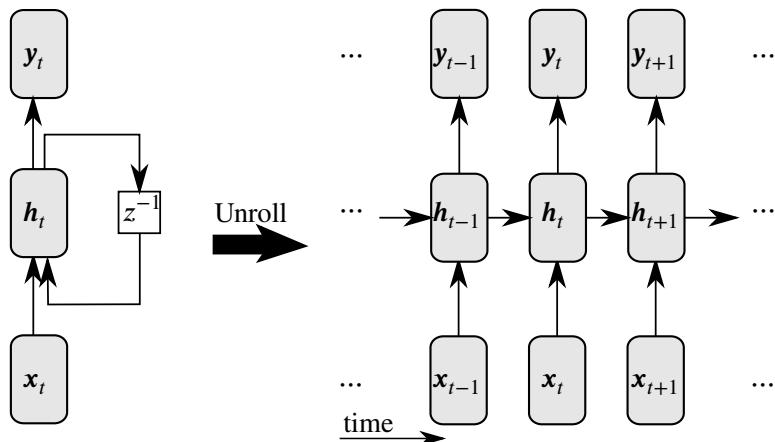


Fig. 2.10 Signal flow diagram representation of a single-layer RNN and its corresponding directed acyclic graph after unrolling

⁹ ?? makes it obvious how the hidden state is carried along throughout the sequence of
¹⁰ computations, giving rise to a useful alternative interpretation of the hidden state as a temporal
¹¹ memory mechanism. Under this interpretation, we can view the hidden state update $\mathbf{h}_t =$
¹² $f_h(\mathbf{x}_t, \mathbf{h}_{t-1})$ as *writing* information extracted from the current inputs \mathbf{x}_t to the memory \mathbf{h}_{t-1} .
¹³ Similarly, producing the outputs $\mathbf{y}_t = f_y(\mathbf{h}_t)$ can be seen as *reading* information from the
¹⁴ hidden state.

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

Stacking memory cells to form multi-layer RNNs

Since the RNN outputs \mathbf{y} also form a sequence with the same length as the inputs \mathbf{x} , they can be used as inputs into another RNN. This stacking of multiple memory cells is similar to the layering seen in deep neural networks, giving rise to the term *deep neural sequence models*

fliang: Cite?

. This is illustrated in ?? .

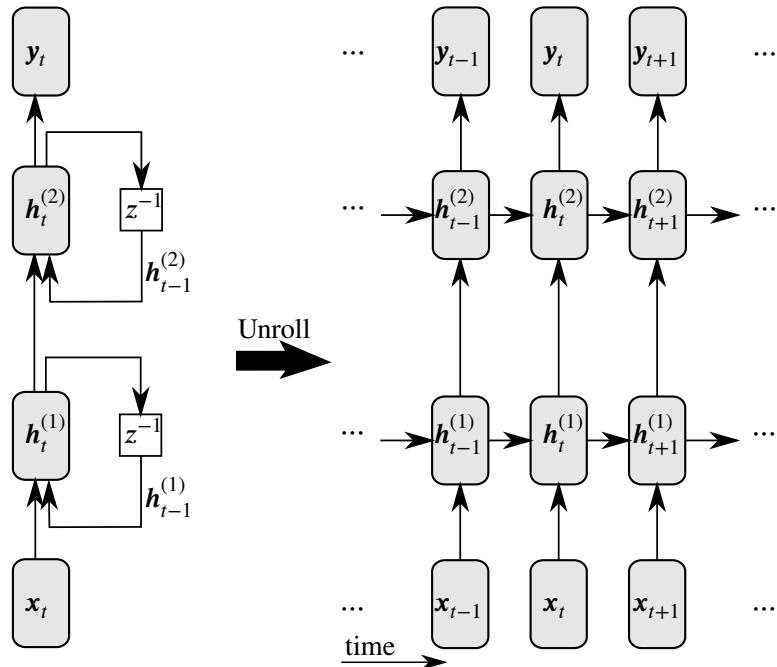


Fig. 2.11 Template and unrolling of a stacked 2-layer RNN

The greater modeling capabilities of multilayer RNNs can be attributed to two primary factors: composition of multiple non-linearities and an increase in the number of paths through which information can flow. The former is analogous to the feedforward case: stacking memory cells increases the number of non-linearities in the composite cell just like stacking multiple layers in feedforward networks. To understand the latter point, notice that in ?? there is only a single path from \mathbf{x}_{t-1} to \mathbf{y}_t hence the conditional independence $\mathbf{y}_t \perp\!\!\!\perp \mathbf{x}_{t-1} | \mathbf{h}_t^{(1)}$ is satisfied. However, in ?? there are multiple paths from \mathbf{x}_{t-1} to \mathbf{y}_t (e.g. passing through either $\mathbf{h}_{t-1}^{(2)} \rightarrow \mathbf{h}_t^{(2)}$ or $\mathbf{h}_{t-1}^{(1)} \rightarrow \mathbf{h}_t^{(1)}$) through which information may flow. Additionally, the hidden state transitions occur on two separate memory cells so parameters need not be tied and the stacked RNN can learn different time dynamics at each depth.

2.2.4 Modeling assumptions

fliang: Integrate better

We make the following assumptions about the sequence $\mathbf{x}_{1:T}$, $\mathbf{y}_{1:T}$, and $\mathbf{h}_{0:T}$:

1. Modified Markov assumption:

$$\forall t : P(\mathbf{h}_t | \mathbf{h}_{0:t-1}, \mathbf{x}_{1:t}) = P(\mathbf{h}_t | \mathbf{h}_{t-1}, \mathbf{x}_t) \quad (2.6)$$

2. Hidden State Stationarity:

$$\forall t_1, t_2 : P(\mathbf{h}_{t_1} = \mathbf{k} | \mathbf{h}_{t_1-1} = \mathbf{i}, \mathbf{x}_{t_1} = \mathbf{j}) = P(\mathbf{h}_{t_2} = \mathbf{k} | \mathbf{h}_{t_2-1} = \mathbf{i}, \mathbf{x}_{t_2} = \mathbf{j}) \quad (2.7)$$

3. Output Stationarity:

$$\forall t_1, t_2 : P(\mathbf{y}_{t_1} = \mathbf{j} | \mathbf{h}_{t_1} = \mathbf{i}) = P(\mathbf{y}_{t_2} = \mathbf{j} | \mathbf{h}_{t_2} = \mathbf{i}) \quad (2.8)$$

4. Output independence:

$$P(\mathbf{y}_{1:T} | \mathbf{h}_{0:T}, \mathbf{x}_{1:T}) = \prod_{t=1}^T P(\mathbf{y}_t | \mathbf{h}_t, \mathbf{x}_t) \quad (2.9)$$

These assumptions imply the sequential factorization:

$$P(\mathbf{y}_{1:T}, \mathbf{h}_{1:T} | \mathbf{h}_0, \mathbf{x}_{1:T}) \quad (2.10)$$

$$= P(\mathbf{y}_{1:T} | \mathbf{h}_{0:T}, \mathbf{x}_{1:T}) P(\mathbf{h}_{1:T} | \mathbf{h}_0, \mathbf{x}_{1:T}) \quad (2.11)$$

$$= \left(\prod_{t=1}^T P(\mathbf{y}_t | \mathbf{h}_t) \right) P(\mathbf{h}_{1:T} | \mathbf{h}_0, \mathbf{x}_{1:T}) \quad ?? \quad (2.12)$$

$$= \left(\prod_{t=1}^T P(\mathbf{y}_t | \mathbf{h}_t) \right) \left(\prod_{t=1}^T P(\mathbf{h}_t | \mathbf{h}_{0:t-1}, \mathbf{x}_{1:t}) \right) \quad (2.13)$$

$$= \left(\prod_{t=1}^T P(\mathbf{y}_t | \mathbf{h}_t) \right) \left(\prod_{t=1}^T P(\mathbf{h}_t | \mathbf{h}_{t-1}, \mathbf{x}_t) \right) \quad ?? \quad (2.14)$$

$$= \prod_{t=1}^T P(\mathbf{y}_t | \mathbf{h}_t, \mathbf{x}_t) P(\mathbf{h}_t | \mathbf{h}_{t-1}, \mathbf{x}_t) \quad (2.15)$$

$$P(\mathbf{y}_{1:T}, \mathbf{h}_{1:T} | \mathbf{h}_0, \mathbf{x}_{1:T}) = \prod_{t=1}^T P(\mathbf{y}_t | \mathbf{h}_t, \mathbf{x}_t) P(\mathbf{h}_t | \mathbf{h}_{t-1}, \mathbf{x}_t) \quad (2.16)$$

2.2 Neural sequence probability modeling

41

?? combined with ?? imply that $P(\mathbf{y}_t|\mathbf{h}_t)$ and $P(\mathbf{h}_t|\mathbf{h}_{t-1}, \mathbf{x}_t)$ are time-invariant and can be modelled by the same recurrent function.

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

$$\mathbf{h}_t = f_h(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad (2.17)$$

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

$$P(\mathbf{y}_{1:T}|\mathbf{h}_0, \mathbf{x}_{1:T}) = P(\mathbf{y}_{1:T}, \mathbf{h}_{1:T}|\mathbf{h}_0, \mathbf{x}_{1:T}) = \prod_{t=1}^T P(\mathbf{y}_t|\mathbf{h}_t, \mathbf{x}_t) f_h(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad (2.18)$$

fliang: Draw PGM

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

fliang: ref

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

fliang: Define V to be the vocabulary

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

Fortunately, we can treat \mathbf{z}_t as the *scores* for a *Boltzmann distribution* (aka softmax):

$$P(\mathbf{y}_t = s|\mathbf{h}_{t-1}, \mathbf{x}_t) = \frac{\exp(-\mathbf{z}_{t,s}/T)}{\sum_{k=1}^K (\exp -\mathbf{z}_{t,k}/T)} \quad (2.19)$$

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

Note the similarity between ?? – ?? and the assumptions for Hidden Markov models [81]. Discrepancies are due to the presence of an input sequence $\mathbf{x}_{1:T}$ in our sequence-to-sequence model.

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

2.2.5 Training RNNs: backpropogation through time

The parameters θ of a RNN are typically learned from data to minimize a cost $\mathcal{E} = \sum_{1 \leq t \leq T} \mathcal{E}_t(\mathbf{x}_t)$ measuring the performance of the network on some task. This optimization is commonly car-

1 tried out using iterative gradient descent methods, which require computation of the gradients
 2 $\frac{\partial \mathcal{E}}{\partial \theta}$ at each iteration.

3 One approach for computing the necessary gradients is *backpropagation through time*
 4 (BPTT)[44], an adaptation of the backpropagation algorithm[63, 85] to the unrolled RNN
 5 computation graph. We can apply the chain rule to the unrolled RNN’s computation graph
 6 in ?? to obtain

$$7 \quad \frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{E}_t}{\partial \theta} \quad (2.20)$$

$$8 \quad \frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left(\frac{\partial \mathcal{E}_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial \theta} \right) \quad (2.21)$$

$$9 \quad \frac{\partial h_t}{\partial h_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} W_{hh}^\top \text{diag}(\sigma'_{hh}(h_{i-1})) \quad (2.22)$$

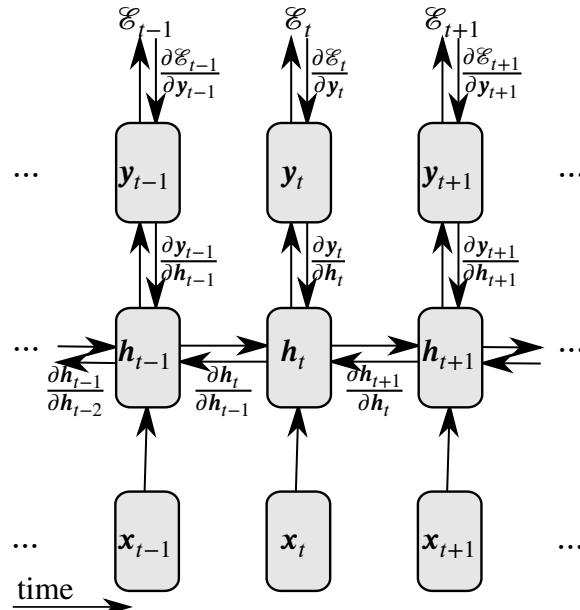


Fig. 2.12 The gradients passed along network edges during BPTT.

10 ?? expresses how the error \mathcal{E}_t at time t is a sum of *temporal contributions* $\frac{\partial \mathcal{E}_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial \theta}$
 11 measuring how θ ’s impact on h_k affects the cost at some future time $t > k$. The factors in
 12 ?? measures the affect of the hidden state h_k on some future state h_t where $t > k$ and can be
 13 interpreted as transferring the error “in time” from step t back to step k [77].

Vanishing/exploding gradients

Unfortunately, naive implementations of ?? and ?? oftentimes suffers from two well known problems: the *vanishing gradient* and *exploding gradient*[10]. Broadly speaking, these problems are both related to the product in ?? exponentially growing or shrinking for long timespans (i.e. $t \gg k$).

Following Pascanu et al. [77], 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

$$\|A\| = \sup_{x \in \mathbb{R}^n; x \neq 0} \frac{|Ax|}{|x|} \quad (2.23)$$

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

From submultiplicativity, we have that for any k

$$\left\| \frac{\partial \mathbf{h}_k}{\partial \mathbf{h}_{k-1}} \right\| \leq \|\mathbf{W}_{hh}^\top\| \|\text{diag}(\sigma'_{hh}(\mathbf{h}_{k-1}))\| \leq \gamma_{\mathbf{W}} \gamma_{\sigma} \quad (2.24)$$

where we have defined $\gamma_{\mathbf{W}} = \|\mathbf{W}_{hh}^\top\|$ and

$$\gamma_{\sigma} := \sup_{\mathbf{h} \in \mathbb{R}^n} \|\text{diag}(\sigma'_{hh}(\mathbf{h}))\| \quad (2.25)$$

$$= \sup_{\mathbf{h} \in \mathbb{R}^n} \max_i \sigma'_{hh}(\mathbf{h})_i \quad \text{Operator norm of diag} \quad (2.26)$$

$$= \sup_{x \in \mathbb{R}} \sigma'_{hh}(x) \quad \sigma_{hh} \text{ acts elementwise} \quad (2.27)$$

Substituting back into ??, we find that

$$\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \right\| = \left\| \prod_{i \geq k} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \right\| \leq \prod_{i \geq k} \left\| \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \right\| \leq (\gamma_{\mathbf{W}} \gamma_{\sigma})^{t-k} \quad (2.28)$$

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

If γ_{σ} is bounded, sufficient conditions for vanishing gradients to occur may be written as

$$\gamma_{\mathbf{W}} < \frac{1}{\gamma_{\sigma}} \quad (2.29)$$

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

¹ The converse of the proof implies that $\|\mathbf{W}_{hh}^\top\| \geq \frac{1}{\gamma_\sigma}$ are necessary conditions for $\gamma_{\mathbf{W}}\gamma_\sigma > 1$
² and exploding gradients to occur.

³ 2.2.6 Long short term memory: solving the vanishing gradient

⁴ In order to build a model which learns long range dependencies, vanishing gradients must be
⁵ avoided. This requires us to design our memory cells holding the hidden state \mathbf{h} such that
⁶ ?? does not hold.

⁷ The *long short term memory* (LSTM) memory cell was proposed by Hochreiter and Schmid-
⁸ huber [54] as a method for dealing with the vanishing gradient problem. It does so by enforcing
⁹ *constant error flow* on ?? , that is

$$\mathbf{W}_{hh}^\top \sigma'_{hh}(\mathbf{h}_t) = \mathbf{I} \quad (2.30)$$

¹¹ where \mathbf{I} is the identity matrix.

¹² Integrating the above differential equation yields $\mathbf{W}_{hh}\sigma_{hh}(\mathbf{h}_t) = \mathbf{h}_t$. Since this should hold
¹³ for any hidden state \mathbf{h}_t , this means that:

¹⁴ 1. \mathbf{W}_{hh} must be full rank

¹⁵ 2. σ_{hh} must be linear

¹⁶ 3. $\mathbf{W}_{hh} \circ \sigma_{hh} = \mathbf{I}$

¹⁷ In the *constant error carousel* (CEC), this is ensured by setting $\sigma_{hh} = \mathbf{W}_{hh} = \mathbf{I}$. This may
¹⁸ be interpreted as removing time dynamics on \mathbf{h} in order to permit error signals to be transferred
¹⁹ backwards in time (??) without modification (i.e. $\forall t \geq k : \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \mathbf{I}$).

²⁰ In addition to using a CEC, a LSTM introduces three gates controlling access to the CEC:

²¹ **Input gate** : scales input \mathbf{x}_t elementwise by $i_t \in [0, 1]$, writes to \mathbf{h}_t

²² **Output gate** : scales output \mathbf{y}_t elementwise by $o_t \in [0, 1]$, reads from \mathbf{h}_t

²³ **Forget gate** : scales previous cell value \mathbf{h}_{t-1} by $f_t \in [0, 1]$, resets \mathbf{h}_t

²⁴ Mathematically, the LSTM model is defined by the following set of equations:

$$i_t = \text{sigmoid}(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{yi}\mathbf{y}_{t-1} + \mathbf{b}_i) \quad (2.31)$$

$$o_t = \text{sigmoid}(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{yo}\mathbf{y}_{t-1} + \mathbf{b}_o) \quad (2.32)$$

$$f_t = \text{sigmoid}(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{yf}\mathbf{y}_{t-1} + \mathbf{b}_f) \quad (2.33)$$

$$\mathbf{h}_t = f_t \odot \mathbf{h}_{t-1} + i_t \odot \tanh(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{yh}\mathbf{y}_{t-1} + \mathbf{b}_h) \quad (2.34)$$

$$\mathbf{y}_t = o_t \odot \tanh(\mathbf{h}_t) \quad (2.35)$$

2.2 Neural sequence probability modeling

where \odot denotes elementwise multiplication of vectors.

Notice that the gates (i_t , o_t , and f_t) controlling flow in and out of the CEC are all time varying. This can be interpreted as a mechanism enabling LSTM to explicitly learn which error signals to trap in the CEC and when to release them [54], enabling error signals to potentially be transported across long time lags.

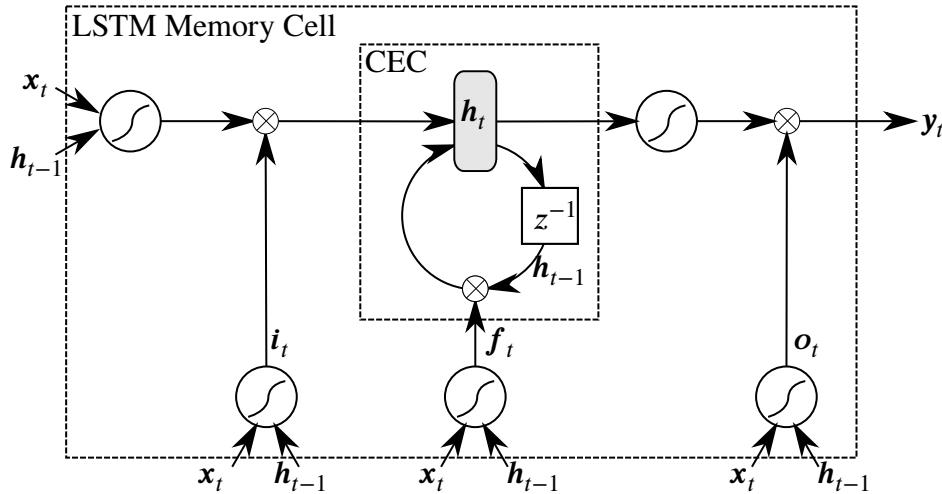


Fig. 2.13 Schematic for a single LSTM memory cell. Notice how the gates i_t , o_t , and f_t control access to the constant error carousel (CEC).

Some authors define LSTMs such that h_t is not used to compute gate activations, referring to the case where h_t is connected as “peephole connections”[41]. We will use LSTM to refer to the system of equations as written above.

Practicalities for successful applications of LSTMs

Many applications of LSTMs

fliang: cite examples

share some common practical techniques for ensuring successful training. Perhaps most important is *gradient norm clipping* [68, 77] where the gradient is rescaled whenever it exceeds a threshold. This is necessary because while vanishing gradients are mitigated by the use of CECs, LSTMs do not explicitly protect against exploding gradients.

Another common practice is the use of methods for reducing overfitting and improving generalization. In particular, *dropout* [88] can be applied to the connections between memory cells in a stacked RNN to regularize the learned features to be more robust to noise [108]. Additionally, *batch normalization*[57] can also be applied to the memory cell hidden states to reduce co-variate shifts, accelerate training, and improve generalization.

Finally, applications of RNNs to long sequences can incur a prohibitively high cost for a single parameter update[[Sutskever](#)]. 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 feedforward network. This issue is typically addressed by only back-propogating error signals a fixed number of timesteps back in the unrolled network, a technique known as *truncated BPTT*[[103](#)]. As the hidden states in the unrolled network have nevertheless been exposed to many timesteps, learning of long range structure is still possible.

3

Related Work

Some common themes

1. Use of domain-specific representations for musical data

2. Modeling at multiple resolutions / timescales (i.e. chords vs notes)

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

3.1 Machine learning on musical data

Computational methods applied to large corpora of music was first described in [18], which termed the phrase “computational musicology.” Since then, development modern tools have greatly aided research efforts. `music21` [20] is a Python programming environment for performing computations over musical composition data which has been utilized for a variety of computational musicology tasks ranging from hierarchical modelling of metrical structure [5], feature generation for downstream machine learning[21], and style classification [51].

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

- 1 1. Classification: the style, composer, or other musical attribute is to be classified
 - 2 2. harmonization: a melody is given and the remaining parts are to be generated
 - 3 3. Completion: given the beginning of a score, the remainder is to be generated
 - 4 4. Automatic Composition: a complete unconstrained score is to be generated
- 5 There is a vast body of research dealing with music classification tasks, including: style
6 classification [51] [23], automated harmonic analysis [74], information retrieval [67], and per-
7 former identification [89]. However, it is not straightforward to utilize work in this area to
8 solve our research goals of music synthesis

9 fliang: of what?

10 .

11 **3.2 Models for music synthesis**

12 Unlike classification, the other tasks (harmonization, completion, automatic composition) re-
13 quire synthesis of novel music. One broad classification of music synthesis methods [98] dis-
14 tinguishes between symbolic (i.e. rule based) versus connectionist (i.e. neural networks).

15 **3.2.1 Symbolic rule-based methods**

16 CHORAL [27] is one of the first rule based expert system for harmonising Bach chorales.
17 It uses 350 manually defined rules as well as hand-tuned search heuristics. A later system
18 called *Experiments in Music Intelligence* (EMI) [17] automatically extracted rules to build an
19 augmented transition network[101]. In [19], grammatical inference is used to learn regular
20 grammars over chord progressions for modelling musical style. [99] applies constraint logic
21 programming for generating harmonizations which satisfy certain harmonic constraints.

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

3.2.2 Connectionist methods

Neural networks have been previously applied to music with varying degrees of success[48]. The earliest connectionist music models utilized note-level Jordan RNNs on melody generation and harmonization tasks [96] [97] [11].

fliang: say more

A landmark connectionist system is Mozer’s CONCERT [71], a BPTT RNN for note-by-note composition. CONCERT models music at two levels of resolution: notes and chords. Notes utilize a psychologically-based representation [86] and chords use a distributed embedding originally trained for style classification [61]. The model passes objective evaluations by faithfully reproducing scales but “while the local contours made sense, the pieces were not musically coherent, lacking thematic structure and having minimal phrase structure and rhythmic organisation“ [71].

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

[66] extended the RNN-RBM[12] to use a LSTM instead of a RNN for modelling hidden unit time dynamics. Unfortunately, the authors do not evaluate their model beyond stating “LSTM-RBM could learn melody lines...while RNN-RBM generates inconsistent and unpleasant sample sequences.”

3.2.3 Hybrid methods

Hybrid approaches which combine both rule-based and connectionist methods have also been investigated. One of the first hybrid systems for music synthesis is HARMONET [52], which combines connectionist neural networks with formal rules to specifically harmonize Bach chorales. It implements a domain-specific processing pipeline consisting of:

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

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

3 fliang: Define this task

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

6 MELONET [32] builds on top of HARMONET’s harmonic modelling to construct chorale
7 partitas (i.e. variations where one of the parts is varied in a harmonically believable way). It
8 first introduced multiple ideas which have been rediscovered in recent years, including:

9 1. Delayed update units to model multiple timescales (described again in Clockwork RNNs
10 [59])

11 2. Use of Resilient Propagation (RProp) [83] for training (described again in [64])

12 Additionally, MELONET utilizes a motif classification neural network to explicitly force mo-
13 tifs to appear multiple times within a partita. Follow up work [56] extends MELONET to
14 use a distributed representation for motifs and a genetic algorithm for training. While MEL-
15 ONET introduces many novel ideas, its limited training set size (16 Pachelbel chorales [55])
16 and domain-specific architecture limit the generalizability of results.

17 fliang: Better analyze this

18 CHIME [33] adopted the Jordan RNN from [97] to add a second training phase using actor-
19 critic reinforcement learning [94]. The critic is constructed using a collection of “music rules,”
20 enabling incorporation of prior knowledge.

21 3.2.4 LSTM music synthesis models

22 Prior work has demonstrated LSTM possesses many properties desirable for music applica-
23 tions. Their superiority over traditional RNNs has been well documented[40]. They can learn
24 to count and measure time intervals between events spaced arbitrarily far apart in time [41],
25 a property N -gram language models do not possess. [43] demonstrated LSTM learning to
26 produce self-sustaining oscillations at a regular frequency, suggesting that they are capable of
27 discovering periodic structure. [37] evaluates various RNN architectures on variety of music
28 tasks and concludes: “while we have found a task that challenges a single LSTM network, we
29 do not believe that any other recurrent networks we have used would be able to learn these
30 songs.”

31 One of the first applications of LSTMs to music was to improvise blues melody lines
32 [29][30]. Using a LSTM to model blues chord progressions and another to model melody
33 lines given chords, the authors reported that LSTM can learn long term music structure such

as repeated motifs can be learned without explicit modelling (e.g. MELONET). However, the music representation quantized to eighth notes, used considered pitch classes without accounting for octaves, and limited the model to 12 possible chords. Additionally, there was “no explicit way to determine when a note ends,” prohibiting discrimination between four consecutive articulations of a note at the same pitch from a single note held for four timesteps. In contrast, our model accounts for the octaves in addition to pitch class, does not restrict the possible chords, operates at twice the time resolution, and also models when a note ends.

More recently, [91] [92] trained character-level LSTMs on 23,000 folk music scores represented in ABC notation[2], a high-level text format for music. ABC format is unsatisfactory for our use case because polyphonic scores are encoded one part at a time so notes sounding close together in time may appear very far apart in the sequence. As a result, it is unsurprising that the authors do not explicitly address polyphony and present exclusively monophonic results.

Many variants of the LSTM architecture have been proposed. Perhaps the most well known is the gated recurrent unit (GRU)[15], which constrains the input and forget gates to sum to 1. [69] proposed the structurally constrained RNN (SCRN), a simple architecture achieving comparable performance to LSTMs. Of most relevance to music, [59] proposed the clockwork RNN for explicitly modelling phenomena occurring at multiple timescales by updating different blocks of the hidden state at different periods. Whether these differences matter is not definitive: [47] performed 5400 experiments on eight different architectures and found no significant difference in performance compared to the original LSTM architecture. [73] reports LSTMs significantly outperform GRUs in music applications.

3.3 Generative modelling of Bach Chorales

One of the first generative models for harmonizing Bach chorales is Bellgard and Tsai’s effective Boltzmann machine model [6]. Their model uses Boltzmann machines to enforce consistency within local contexts. As a result, their model is unable to capture long-range dependencies. Furthermore, they quantize to half-notes and only achieve 1/8 the time resolution of our model.

[3] used HMMs to harmonize Bach chorales. Their model consists of two separate HMM models: one for generating harmonizations and another for ornamentation. Their model uses a discrete harmonic encoding of chords for hidden states. In contrast, our model uses an unconstrained continuous hidden state and requires no separate ornamentation step.

[64] applied LSTMs to Bach chorales and reports significant gains using RProp instead of BPTT, a technique previously utilized by MELONET[32]. However, they erroneously use a

¹ mean squared error training criterion for a classification task, casting doubts on the validity of
² their experiments.

³ [13] compared RNN models for Bach chorales and found clockwork RNNs to yield the
⁴ lowest validation loss. However, their data format does not permit independent articulation of
⁵ parts. More importantly, the performance margin between clockwork RNNs and LSTMs was
⁶ very small (6.5 vs 6.75 cross-entropy loss) and their implementation resets the LSTM state
⁷ when truncating gradients during BPTT, limiting the time-range of learned dynamics to be at
⁸ most the sequence length.

⁹ fliang: Tom Collins work

4

Automatic composition

fliang: Debug minitoc

This chapter describes a generative RNN sequence model for polyphonic music. We first describe our process for constructing a training corpus of chorales and quantify the impact of our preprocessing procedure over the corpus. Next, we present a simple frame-based sequence encoding for polyphonic music. Using this sequence representation, we perform an investigation of various RNN architectures, design tradeoffs, and training methods. The results of our investigation are used to design our final model, which we compare against prior work and utilize in following chapters.

4.1 Constructing a corpus of encoded 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 [80] (i.e. no parallel fifths) and stylistic elements (i.e. voice leading) which can be used to qualitatively evaluate success
2. The structure of chorales are regular: all chorales have four parts and consist of a melody in the Soprano part harmonized by the Alto, Tenor, and Bass parts. Additionally, each

1 chorale consists of a series of *phrases*: groupings of consecutive notes into a unit that
2 has complete musical sense of its own[72]. It is well known

3 fliang: citep

4 that Bach denoted ends of phrases with fermatas

5 fliang: refer back to background

6 .
7 3. The Bach chorales have become a standardized corpus routinely studied by aspiring mu-
8 sic theorists[102]

9 The Bach chorales, indexed by the Bach-Werke-Verzeichnis (BWV) numbering system[14],
10 are conveniently provided by music21[20].

11 4.1.1 Preprocessing

12 Motivated by the transposition invariance of music and prior practice [71] [29] [35] [36], we
13 first perform *key normalization*. The key signature of each score were firsanalyzed using the
14 Krumhansl Schmuckler key-finding algorithm [60] and then transposed according to

15 fliang: Table XYZ

16 such that the transposed key is C-major for major mode scores and A-minor for minor mode
17 scores.

18 Next, we perform *time quantization* by aligning note start and end times to the nearest
19 multiple of some minimum duration. Our model uses a minimum duration of one 1/16th note,
20 exceeding the time resoltuions of [12] [29] by 2x, [52] by 4x, and [6] by 8x.

21 We consider only note pitches and durations, neglecting changes in timing (e.g. ritardan-
22 dos), dynamics (e.g. crescendos), and stylistic notations (e.g. accents, staccatos, legatos).

23 An example of the distortion introduced through of our preprocessing steps is provided
24 in 4.1 on the facing page in sheet music notation and in piano roll

25 fliang: Is piano roll defined?

26 notation on 4.2 on page 63.

27 Corpus level analysis of preprocessing effects

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

4.1 Constructing a corpus of encoded scores

The image displays two staves of musical notation for a four-part chorale. The top staff is in E-flat major (B-flat bass) and the bottom staff is in C major (F bass). Both staves show four bars of music with various note heads and stems.

Fig. 4.1 First 4 bars of JCB Chorale BWV 133.6 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.

¹ ?? plots a histogram of pitch usage counts before and after key normalization. Notice that
² the overall range of pitches has increased after key normalization. This can be explained by
³ noting that Bach’s chorales were to be performed by vocalists and hence were restricted to use
⁴ pitches within human voice ranges regardless of key. After transposition, this constraint is no
⁵ longer be satisfied and we see the appearance of unrealistically low notes (e.g. A1) outside the
⁶ range of even the lowest voice types.

⁷ In ?? , we visualize histograms of pitch class usages. As expected, key normalization has
⁸ increased the usage of pitch classes in the key of C-major / A-minor (i.e. those which possess
⁹ no accidentals) and decreased out of key pitch classes (e.g. C#, F#).

¹⁰ We investigate the effects of time quantization in ?? , which shows histograms of note
¹¹ duration usages before and after quantization.

¹² fliang: Update plots... are they affected

¹³ .

¹⁴ 4.1.2 Sequential encoding of musical data

¹⁵ After preprocessing of the scores, our next step is to encode music into a format amenable
¹⁶ for sequential processing. Similar to [97], we represent polyphonic scores using a localist
¹⁷ frame-based representation where time is discretized into constant timestep *frames*. Frame
¹⁸ based processing forces the network to learn the relative duration of notes, a counting and
¹⁹ timing task which [43] demonstred LSTM is capable of. Consecutive frames are separated by
²⁰ a unique delimiter (“|||” in

²¹ fliang: Figure of score encoded in text

²²).

²³ Each frame consists of a sequence of $\langle \text{Note}, \text{Tie} \rangle$ tuples where $\text{Note} \in \{0, 1, \dots, 127\}$ rep-
²⁴ resents the MIDI pitch of a note and $\text{Tie} \in \{T, F\}$ distinguishes whether a note is tied with a
²⁵ note at the same pitch from the previous frame or is articulated at the current timestep. A de-
²⁶ sign decision here is the order in which notes within a frame are encoded and consequentially
²⁷ processed by a sequential model. Since chorale music places the melody in the Soprano part, it
²⁸ is reasonable to expect the Soprano notes to be most significant in determining the other parts.
²⁹ Hence, we choose to process the Soprano notes first and order notes in descending pitch within
³⁰ every frame.

³¹ The above specification describes our initial encoding. Later in our work

³² fliang: reference

³³ , we found that this encoding resulted in unrealistically long phrase lengths. Including
³⁴ fermatas (represented by “(.)” in

4.2 Design and validation of a generative model for music

57

fliang: Figure of encoded score

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

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

fliang: Figure

) and end symbol (“END” in

fliang: Figure

) are appended to the beginning and end respectively. This causes the model to learn to initialize itself when given the start symbol and allows us to determine when a composition generated by the model has concluded.

Observe that our encoding is sparse: unarticulated notes are not encoded. It is also variable length as anywhere from zero to four (in the case of chorales, more for arbitrary polyphonic scores) notes. Finally, the explicit representation of tied notes vs articulated notes solves the problem plaguing [29][28] [64] [13] where multiple articulations at the same pitch are indistinguishable from a single note with the same duration.

Additionally, notice that our encoding avoids hand-engineered features such as pitch representations which are psychochologically-based [71] or harmonically-based [35] [61]. This is intentional and is motivated by numerous reports [8][9] suggesting that that a key ingredient in deep learning’s success is its ability to learn good features from raw data.

fliang: Describe the dataset

Total vocabulary size: 108 Total tokens in file: 846926 Training size: 677542+84692 Val size: 84692

4.2 Design and validation of a generative model for music

In this section, we describe the design and validation process leading to our generative model. Unlike many prior models for music data, we intentionally avoid injection of domain-specific knowledge into our model architectures such as distinguishing between chords versus notes [52][71] [29] and explicitly modelling of meter [28] or motifs [32]. Through this fundamental connectionist approach, we aim to minimize biases introduced by prior assumptions and force the model itself to learn musical structure from data.

4.2.1 Training and evaluation criteria

Following [71], we will train the model to predict a distribution distribution over all possible tokens next \mathbf{x}_{t+1} given the current token \mathbf{x}_t and the previous hidden state \mathbf{h}_{t-1} . This is equivalent

1 to setting the target sequence to be the input sequence delayed by one timestep: $\mathbf{y}_{1:T-1} = \mathbf{x}_{2:T}$
 2 and $\mathbf{y}_T = \text{STOP}$.

3 fliang: Diagram for sequential prediction

4 .
 5 fliang: Note similarity with language modeling

6 .
 7 For training criteria, we use the cross-entropy between the predicted distributions $P(\mathbf{y}_t | \mathbf{h}_t, \mathbf{x}_t)$
 8 and the actual target distribution $\delta_{\mathbf{y}_t}$.

9 Note that our training criteria as written in

10 fliang: reference

11 uses the actual next token \mathbf{x}_{t+1} as the recurrent input, even if the most likely prediction
 12 argmax $P(\mathbf{x}_{t+1} | \mathbf{h}_t, \mathbf{x}_t)$ differs. This is referred to as *teacher forcing*[104] and is motivated
 13 by the observation that model predictions may not yet be correct during the early iterations
 14 of training. However, at inference the token generated from $P(\mathbf{x}_{t+1} | \mathbf{h}_t, \mathbf{x}_t)$ is reused as the
 15 previous input, creating a discrepancy between training and inference. Scheduled sampling
 16 [7] is a recently proposed alternative training method for resolving this discrepancy and may
 17 help the model better learn to predict using generated symbols rather than relying on ground
 18 truth to be always provided as input.

19 4.2.2 Comparing memory cells for music data

20 Using theanets

21 One-hot encoding, 64 dimensional vector embedding, RNN layer, fully connected layer,
 22 softmax.

23 ?? compares various RNN architecture performance through training a RNN with num_layers=1,
 24 rnn_size=130, wordvec=64.

25 90/10 train/test split.

26 The LSTM and GRU architectures achieve the lowest training errors, consistent with expectations
 27 since these architectures have the most parameters. All yielded comparable validation
 28 loss which increased over time, motivating regularization.

29 LSTMs and GRUs trained much faster and achieved lower training loss, suggesting higher
 30 capacity. [73] reports that LSTMs outperform GRUs in music applications, motivating our
 31 final choice for GRUs.

4.2.3 Optimizing the LSTM architecture

Switched to `torch-rnn`.

fliang: Discrepancy between above architectures and below losses because we are perturbing about best model

We construct multi-layer LSTM models with `num_layers` number of layers, each containing `rnn_size` hidden units. The inputs x_t are one-hot-encoded before being passed through a wordvec-dimensional vector-space embedding layer, which compresses the dimensionality down from $|V| \approx 140$ to wordvec dimensions. Dropout layers were added between LSTM connections in both depth and time dimensions all with dropout probability $\text{dropout} \in [0, 1]$.

We build our models using the `torch7` framework and an optimized implementation of LSTMs provided by `torch-rnn`

fliang: citep

Models were trained using RMSProp

fliang: citep

with batch normalization

fliang: citep

and an initial learning rate of 2×10^{-3} decayed by 0.5 every 5 epochs. The back-propagation through time gradients were clipped at t

fliang: citep Mikolov

and truncated after `seq_length` time steps. We use a mini-batch size of 50.

Overall best model

We identified our best model to be

fliang: what is it?

In the following sections, we investigate perturbations about this model and the effects of various hyperparameters. A complete listing of results are available in ?? .

Regularization

The increasing validation error in ?? confirmed that our models were overfitting and required regularization. dropout

fliang: Batch normalization experiments

1 **Network capacity**

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

- 3 • Larger `rnn_size` leads to higher capacity and lower training loss
- 4 – Presents as overfitting on validation, where the lowest capacity model `rnn_size`
5 appears to be improving in generalization while others are flat/increasing
- 6 • Training curves about the same wrt `num_layers`, validation curves have interesting story
- 7 – Depth matters: small 64 and 128 hidden unit RNNs saw improvements up to 0.09
- 8 – Expressivity gained from depth furthers overfitting: 256 hidden unit RNN has some
9 of the best validation performance at depth 1 but is the worst generalizing model
10 for depths 2 and 3 even though training loss is low
- 11 • `rnn_size=128` undisputedly best generalizing, optimized at `num_layers=2`: will con-
12 tinue with these settings

13 **Network input parameters**

14 fliang: Is `seq_length` an input parameter, or the BPTT parameters?

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

- 16 • Training losses are about the same across all `wordvecs`
- 17 • Validation losses suggest that increasing `seq_length` important for good performance
- 18 fliang: investigate further
- 19 • `wordvec=128` overfits for all cases, the other two depend on `seq_length` and vary an
20 order of magnitude smaller than the performance gains from increasing `seq_length`

21 **4.3 Results**

22 Allan and Williams [3] achieve cross-entropy losses of 2.79 – 2.80 on unseen test-set scores
23 respectively for their “harmonic skeleton” subtask. This task involves predictiong a sequence
24 of one of 81 harmonic symbols, which may be interpreted as equivalence classes of chords.
25 Even after applying Viterbi decoding to find globally optimal sequences, their HMM models
26 achieve cross entropies of 0.84 – 0.87 on training-set scores (Table 5.2 in Allan and Williams
27 [3]).

4.4 Accelerating model training with GPUs

61

The baseline results obtained by N -grams are shown in 4.1 on the following page. We compare against two widely available language modelling software packages, KenLM [50] and SRILM [90]. KenLM implements an efficient modified Kneser-Ney smoothing language model and while SRILM provides a variety of language models we choose to use the Good-Turing discounted language model for benchmarking against.

In contrast, our best model achieves cross-entropy losses of 0.323 on training data and 0.477 on held-out test data (1 on page 104), corresponding to a training perplexity of 1.251 bits and a test perplexity of 1.391. This is more than 0.6 bits lower than any test perplexity obtained by the N -gram models compared in , providing evidence that the additional modelling capacity provided by RNNs is useful for encoded music score data.

The best models investigated in Boulanger-Lewandowski et al. [12] achieved -5.56 log likelihood and 33.12% accuracy on the symbolic prediction task. Note that the log likelihood is significantly lower than the -0.323 log likelihood implied by our model’s cross-entropy loss. This is because the two results are not comparable as they utilize different input encodings with varying vocabulary sizes.

fiang: Compare [13]

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

4.4 Accelerating model training with GPUs

fiang: Show speedup when training with multi-GPU, selling point is how fast the model trains

GPU: 4762.23s user 852.23s system 99% cpu 1:33:36.14 total Forward / Backward pass took 0.51340484619141 Using 243MB of memory

CPU:

Forward / Backward pass took 5.4034311771393

4.5 Other applications

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

Table 4.1 Perplexities of baseline N -gram language models

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

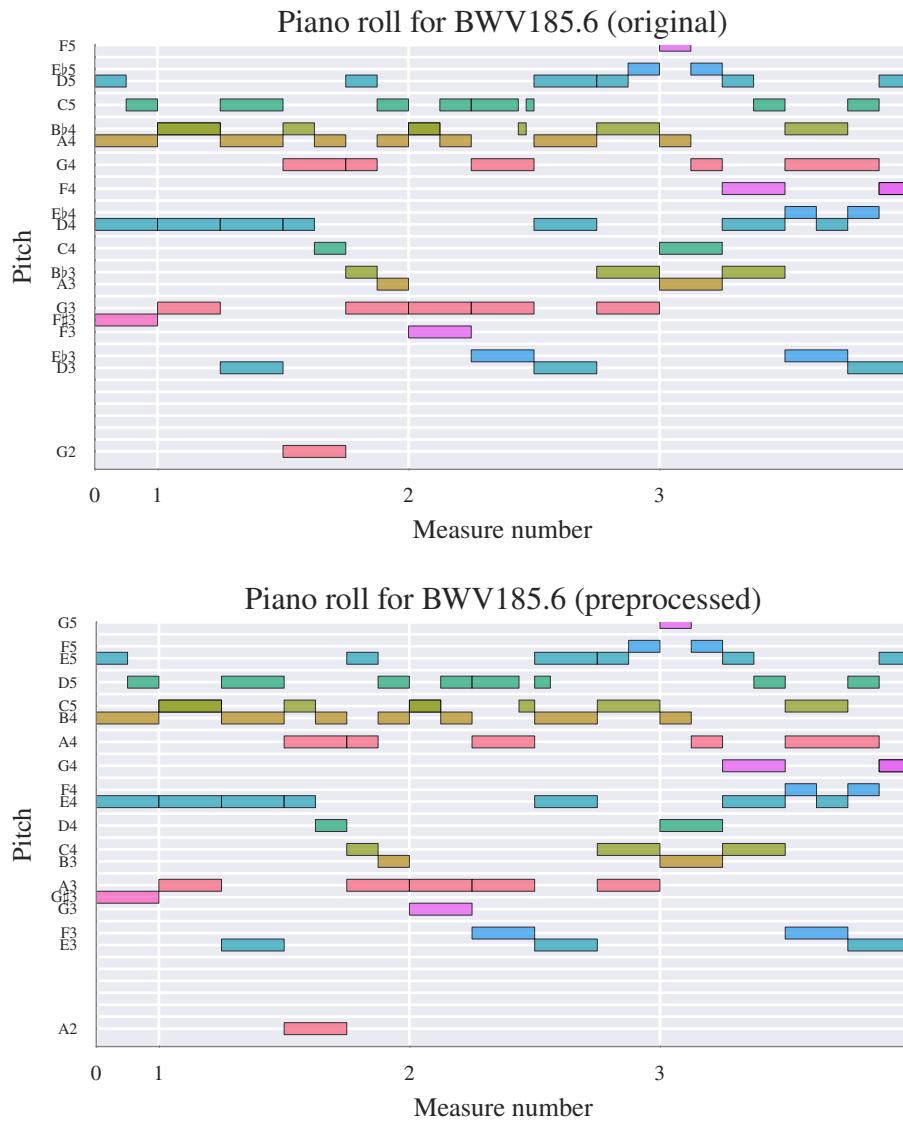


Fig. 4.2 Piano roll representation of the same 4 bars from ?? before and after preprocessing. Notice the transposition of key as well as quantization of the C5 and D5 notes (after transposition) in bar 3 of the Soprano part.

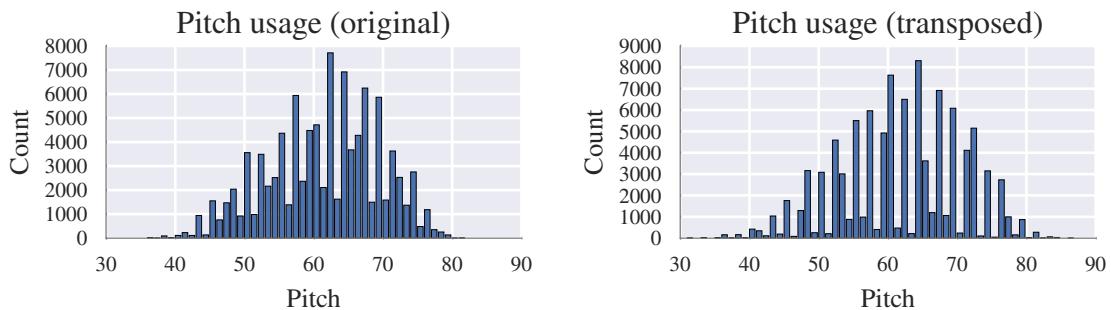


Fig. 4.3 Pitches before and after key standardization

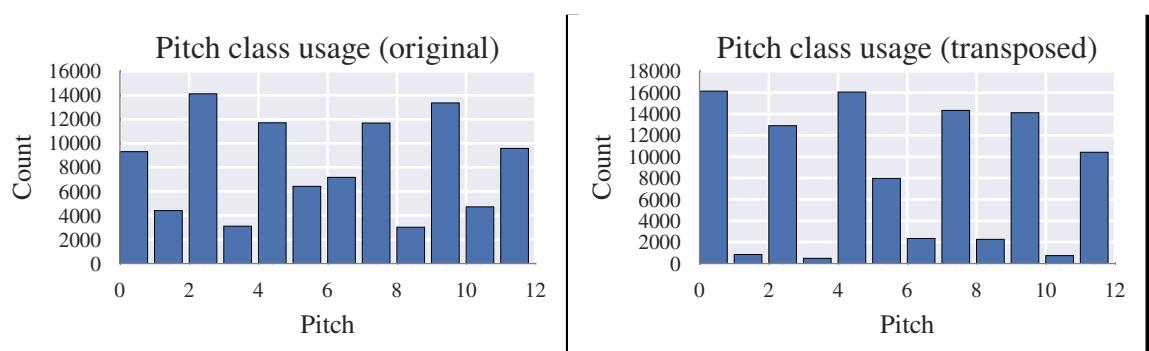


Fig. 4.4 Pitch classes before and after key standardization

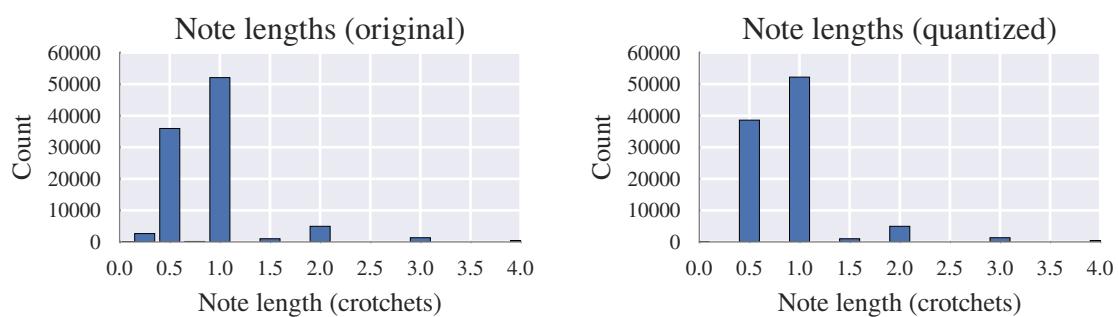


Fig. 4.5 Effects of time quantization on note durations

4.5 Other applications

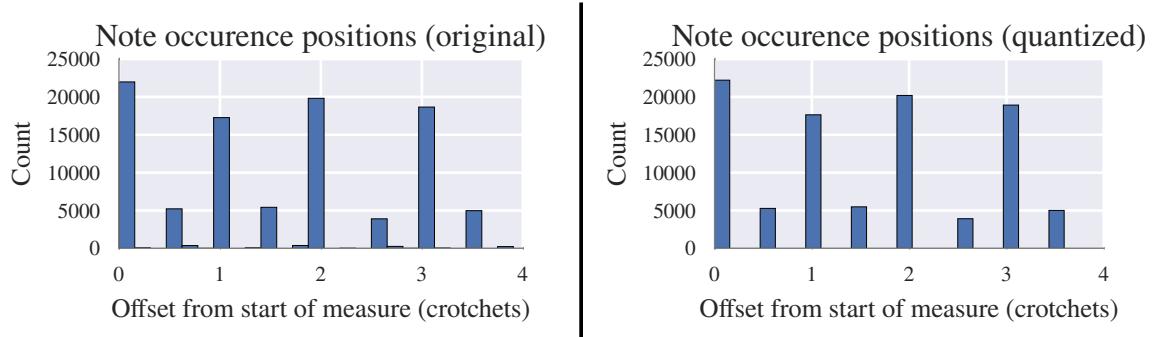


Fig. 4.6 Effects of time quantization on meter

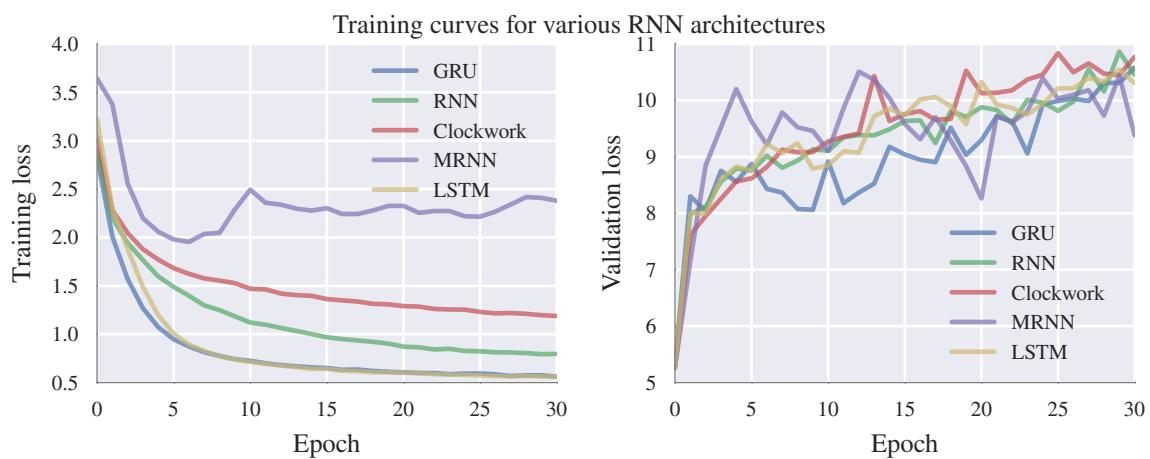


Fig. 4.7 theanets-architecture

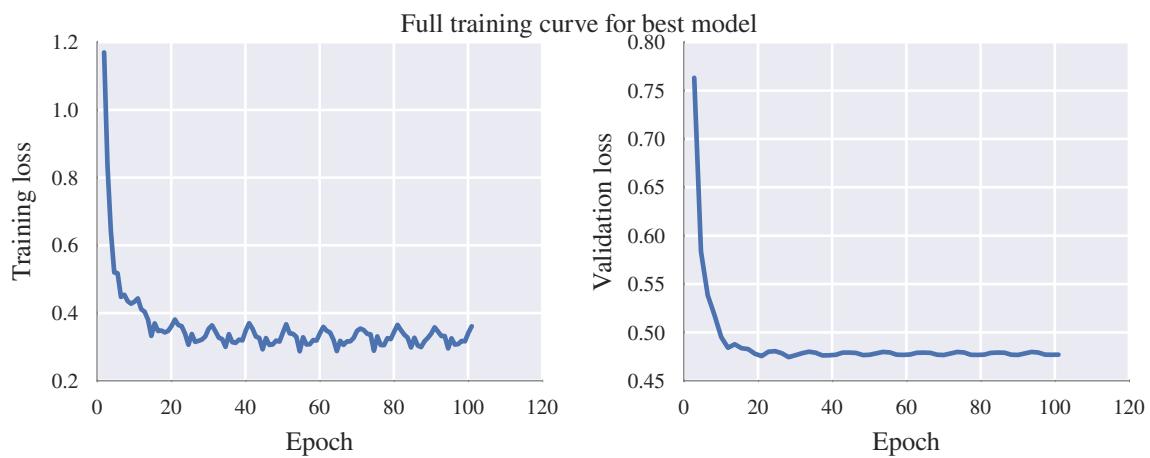


Fig. 4.8 torch-rnn-best-model-Trace

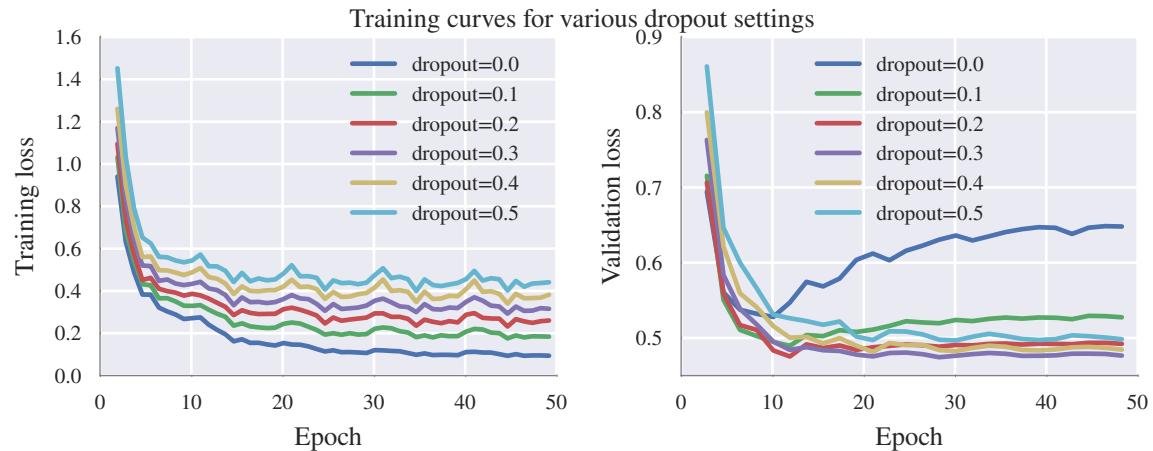


Fig. 4.9 torch-rnn-Dropout

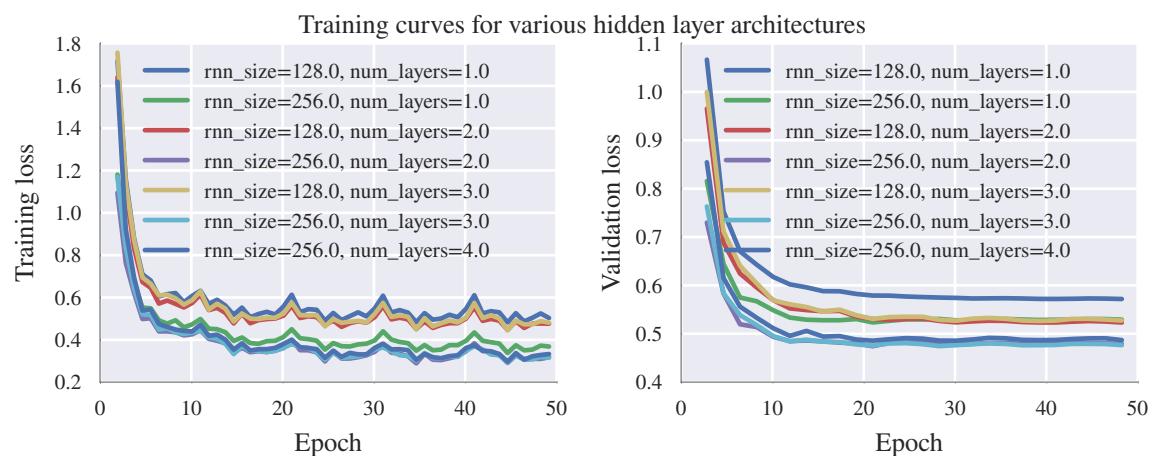


Fig. 4.10 torch-rnn-network-params

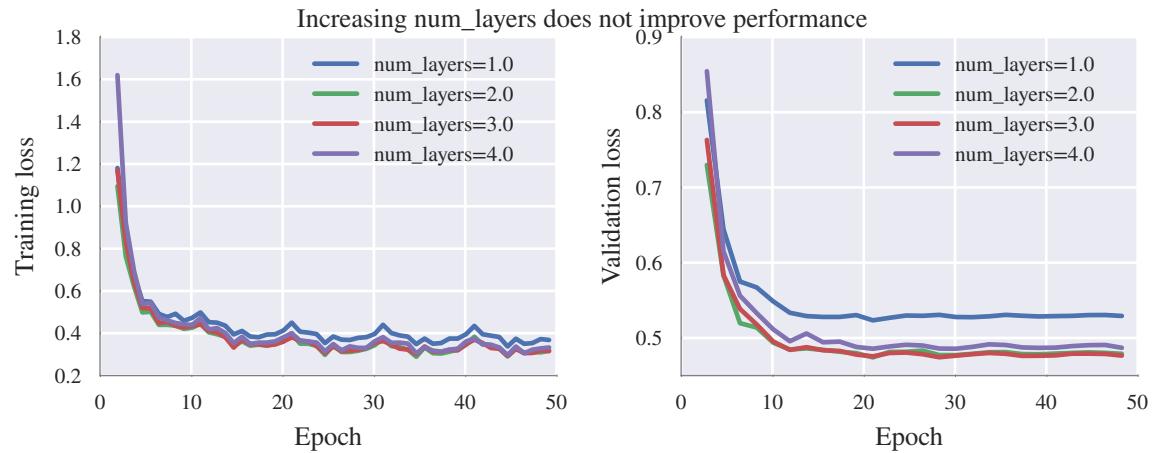


Fig. 4.11 torch-rnn-network-params-num-Layers



Fig. 4.12 torch-rnn-network-params-rnn-size

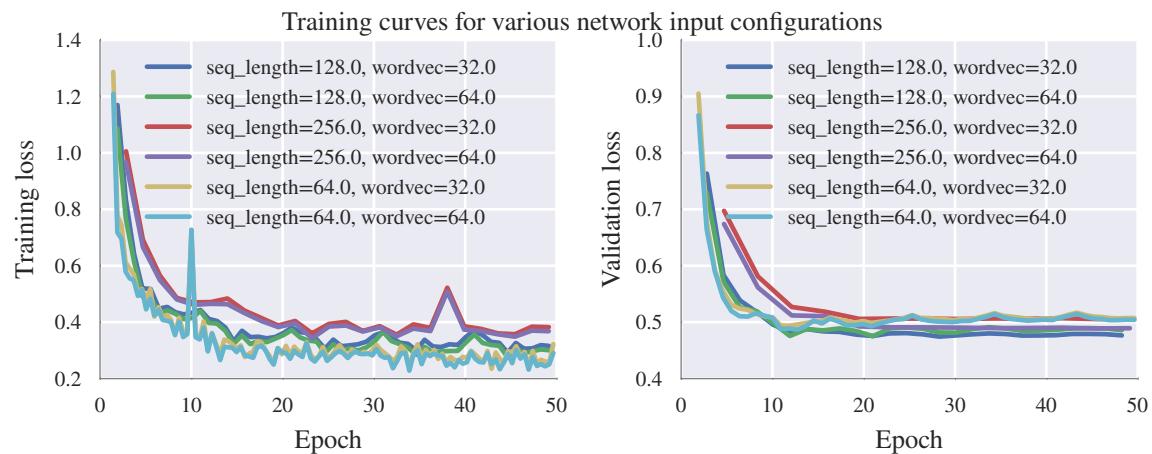


Fig. 4.13 torch-rnn-input-params

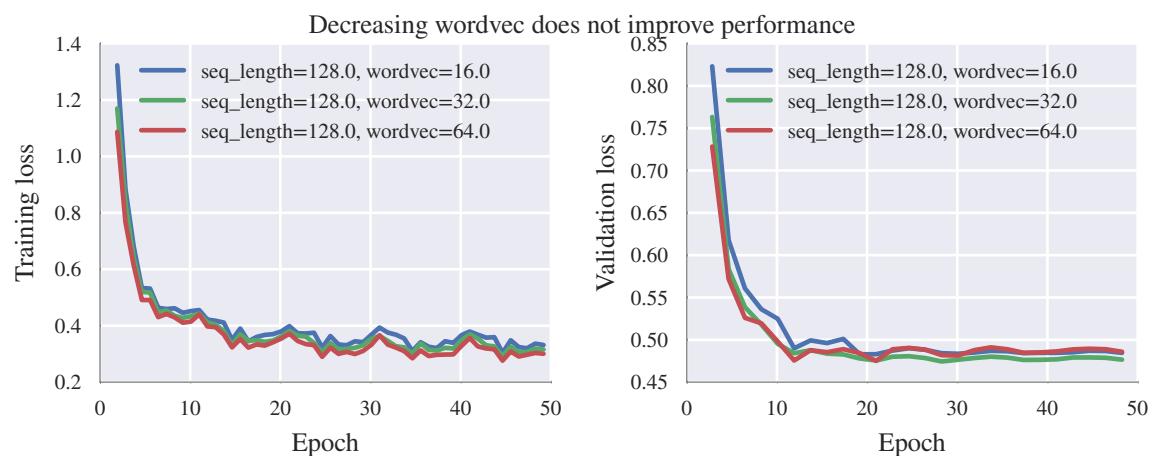


Fig. 4.14 torch-rnn-input-params-Wordvec

5

Chorale harmonization

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

Unlike automatic composition, in harmonization tasks we are given the entire sequence of notes for one or more parts. As one of the parts is now fixed, the model is no longer able to freely compose and harmonic deviations from the fixed parts will result in dissonances and conflicting expectations. This lack of accountability for future expectations is one of the failure modes of our models. One potential method for mitigating this is bidirectional LSTMs[46], which account for both the future and prior contexts. However, a bidirectional LSTM cannot be sequentially sampled to perform automatic composition.

5.1 Background

A chorale consists of four parts: soprano, alto, tenor, and bass. Chorale harmonization involves producing the alto, tenor, and bass parts given a fixed sopran melody. As described by Walter Piston [80]:

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

The Baroque style employed by Bach has specific guidelines such as disallowing parallel fifths and parallel octaves as well as considerations for voice leading [80].

For a music student studying chorale harmonization, a common pedagogical exercise [24][80] is a sequence of tasks increasing in difficulty:

1. Providing either alto and tenor given fixed soprano and bass
2. Providing both alto and tenor parts given fixed soprano and bass
3. Providing all remaining parts given only the soprano line

There are no definitive formalization of the harmonization process, making evaluation difficult. Attempts to formalize the process using Shenkerian structural analysis [75] and symbolic methods such as generative grammars [62][106] exist, but involve human analytical process.

5.2 Harmonizing

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

$$w_{1:L}^* = \underset{w_{1:L}}{\operatorname{argmax}} P(w_{1:L} | w_\alpha) \quad (5.1)$$

“Clamp” the generative model and have it “fill-in” missing bits [53]. We can constrain the set of candidate sequences by first noting any solution $\hat{w}_{1:L}$ must satisfy $\hat{w}_\alpha = w_\alpha$. We can apply this constraint and greedily sample from our generative model to approximately solve the problem:

$$\hat{w}_t = \begin{cases} w_{\alpha_t} & \text{if } t \in \alpha \\ \underset{w_t}{\operatorname{argmax}} \hat{P}(w_t | \hat{w}_{1:t-1}) & \text{otherwise} \end{cases} \quad (5.2)$$

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

This solution is approximate because while the factorization

$$P(w_{1:L}) = \prod_{t=1}^L P(w_t | w_{1:t-1}) \quad (5.3) \quad 2$$

is true and justifies our model, the factorization

$$P(w_{1:L} | w_\alpha) = \prod_{t=1}^L \hat{P}(w_t | w_{1:t-1}) \quad (5.4) \quad 4$$

does not hold. Some primary criticisms include

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

fliang: mikolov

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

Beam search is one way to mitigate the effects of greedy selection: our current method is equivalent to a beam search with width one. Maintaining N -best hypotheses using a lattice-based framework such as in Liu et al. [65] would allow the model to partially recover from mistakes made during greedy action selection.

Furthermore, another potential issue with the proposed model is a discrepancy between the inputs provided during training (which are taken from actual data sequences regardless of model predictions) and sampling (where the inputs are generated by the model at the previous timestep). To resolve this discrepancy, Bengio et al. [7] proposed *scheduled sampling* as a curriculum learning strategy gradually transitioning sequence models to rely on their prior predictions as inputs and thereby learn state dynamics more resilient to erroneous inputs.

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

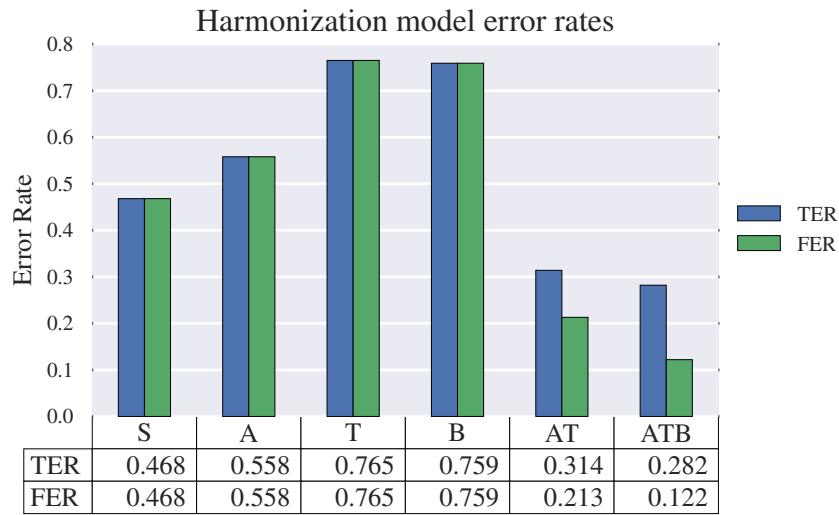


Fig. 5.1 Error rates for harmonization tasks

¹ 5.3 Datasets

² We create datasets where one or more parts are masked:

- ³ • A single voice: Soprano (S), Alto (A), Tenor (T), or Bass (B)
- ⁴ • The middle two voices (AT)
- ⁵ • All voices except Soprano (ATB), aka *harmonization*

⁶ Of particular interest is the AT dataset. Bach oftentimes only wrote the Soprano and Bass
⁷ parts of a piece, leaving the middle parts to be filled in by students. Our networks performance
⁸ on this task can be used as a benchmark for an easier task. Another interesting configuration
⁹ is ATB, which corresponds to harmonizing a given melody (i.e. Soprano line) and can be
¹⁰ compared against prior work such as Allan and Williams [3].

¹¹ 5.4 Results

¹² We are unable to compare against Allan and Williams [3] because their model consists of
¹³ multiple domain specific subtasks (e.g. harmony and chord skeletons, ornamentation) and
¹⁴ because they only consider the ATB case.

5.4 Results

73



Fig. 5.2 Happy birthday soprano melody, ATB harmonized by BachBot

5.4.1 “Bachifying” other music

In addition to being able to harmonize Bach chorales, we found that BachBot was capable of generating Baroque accompaniments to a variety of pieces. ?? shows BachBot’s proposed ATB harmonization for happy birthday

fliang: cite? public domain

and ?? for twinkle twinkle little star

fliang: cite?

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

The musical score consists of two systems of four staves each, representing the voices: Soprano, Alto, Tenor, and Bass. The first system (measures 1-5) has a common time signature and a key signature of one sharp. The second system (measures 6-10) begins at measure 6 with a common time signature and a key signature of one sharp. The vocal parts are harmonized with eighth-note patterns, and the bass part includes some sixteenth-note figures.

Fig. 5.3 Twinkle-twinkle soprano melody, ATB harmonized by BachBot

6

Large-scale subjective evaluation

[79] addresses difficulty in quantitative evaluation, suggesting the use of a learned critic in a manner similar to GANs [45]. In a later report, [78] attribute difficulty in evaluation due to lack of aim: algorithmic composition, design of compositional tools, and computational modelling of musical styles or music cognition all have different motivations and should thus be evaluated differently.

Following advice of [78], we identify our key motivation as algorithmic composition: generation of novel compositions. To evaluate our success, we employ a subjective evaluation method.

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

6.1 Evaluation framework design

6.1.1 Software architecture

The frontend utilizes React and Redux, allowing us to collect fine-grained user action data. Azure App Service is used to host an Express web-service which randomizes experimental



Challenge description

We will present you with some short samples of music which are either extracted from Bach's own work or generated by BachBot. Your task is to listen to both and identify the Bach originals.

To ensure fair comparison, all scores are transposed to C-major or A-minor and set to 120 BPM.

Fig. 6.1 The first page seen by a visitor of <http://bachbot.com>

- ¹ questions and collects responses. The data is stored to Azure Data Storage and processed in
- ² batch MapReduce using Azure HDInsight.

³ 6.1.2 User interface

- ⁴ The landing page for <http://bachbot.com/> is shown in ?? .

⁵ Clicking “Test Yourself” redirects the participant to a user information form (??) where
⁶ users self-report their age group prior music experience into the categories shown.

⁷ After submitting the background form, users were redirected to the question response page
⁸ shown in ?? . This page contains two audio samples, one extracted from Bach and one gen-
⁹ erated by BachBot, and users were asked to select the sample which sounds most similar to
¹⁰ Bach. Users were asked to provide five consecutive answers and then the overall percentage
¹¹ correct was reported.

Some background info about you

Age Group Under 18 18 to 25 26 to 45 46 to 60 Over 60

Self-rating of music experience

- Novice:** I like to listen to music, but do not play any instruments
- Intermediate:** I have played an instrument, but have not studied music composition
- Advanced:** I have studied music composition in a formal setting
- Expert:** I am a teacher or researcher in music

Submit

Clear Values

Fig. 6.2 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 6.1 Composition of questions on <http://bachbot.com>

6.1.3 Question generation

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

fliang: ref

) as well as original compositions (denoted SATB as all parts are generated). For each question, a random chorale was selected without replacement from the corpus and paired with a corresponding harmonization. SATB samples were paired with chorales randomly sampled from the corpus. The five questions 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 ?? for details.

1
2
3
4
5
6
7
8
9
10

The BachBot Challenge

	Select the music most similar to Bach
Select	⏪ 🔊
Select	▶ ⏪
	Submit



Fig. 6.3 Question response interface used for all questions

6.2 Results

6.2.1 Participant backgrounds and demographics

We received a total of

fliang: FILL THIS IN LAST

responses from

fliang: FILL THIS IN LAST

different countries. As evidenced by ?? , our participant is diverse and includes participants

from six different continents. ?? shows that while the majority of our participants are between

18 – 45 and have played an instrument, more than 20%

fliang: FIX NUMBER LAST

have either formally studied or taught music theory.

6.2.2 BachBot's performance results

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

6.2 Results

79

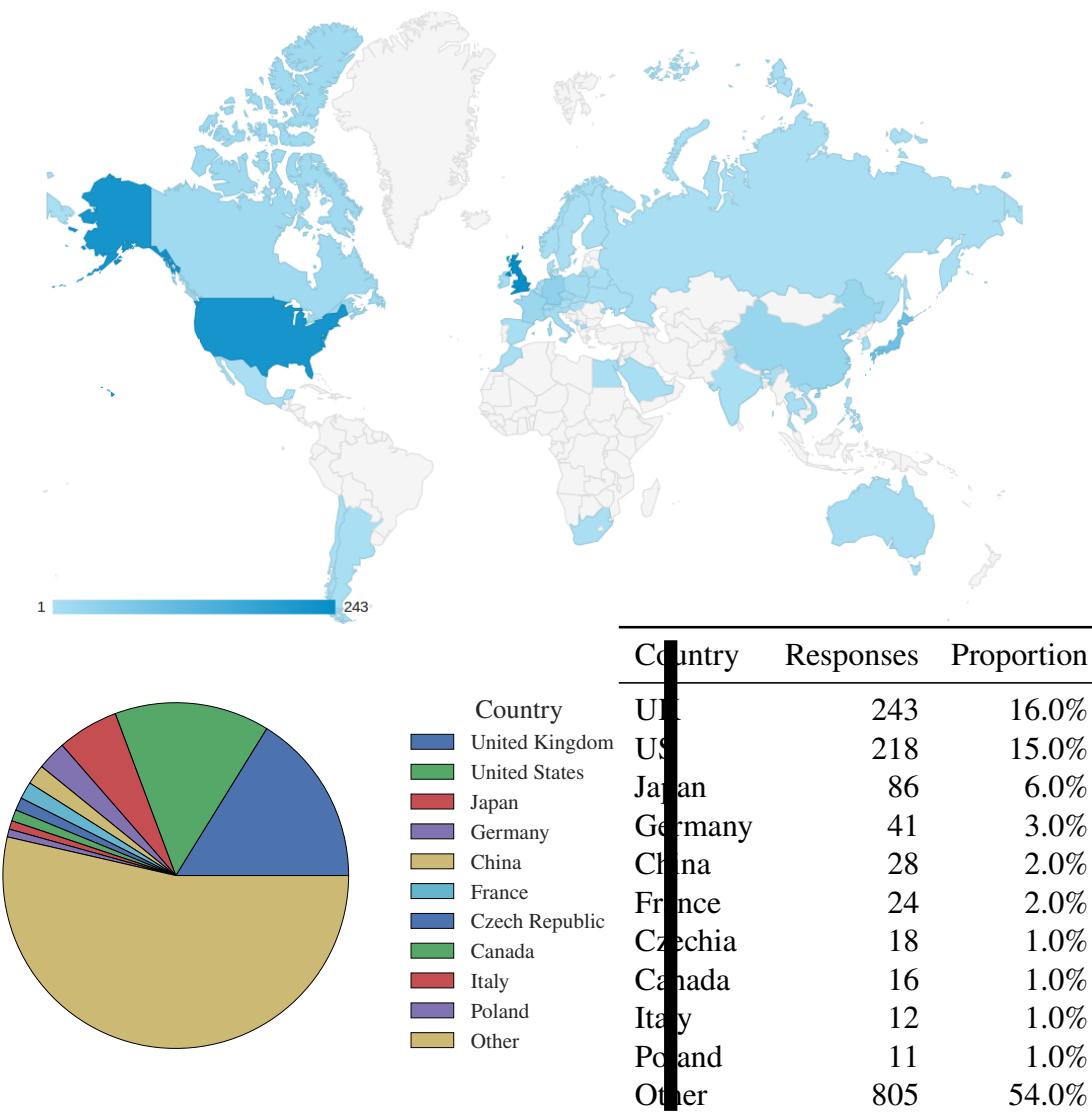


Fig. 6.4 Geographic distribution of participants

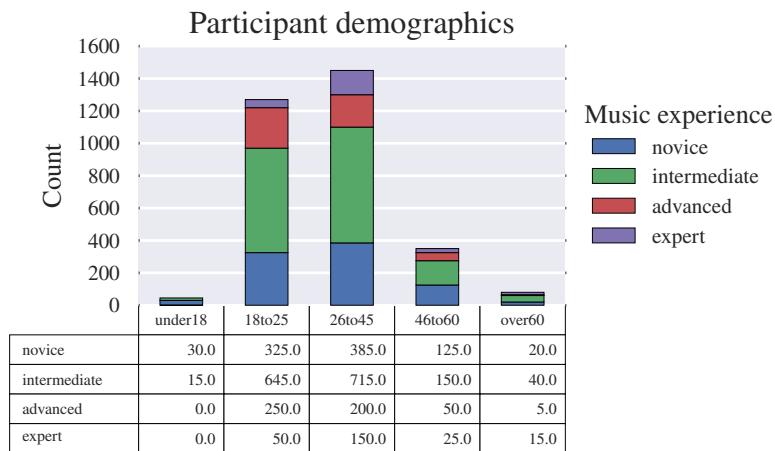


Fig. 6.5 Demographics of participants

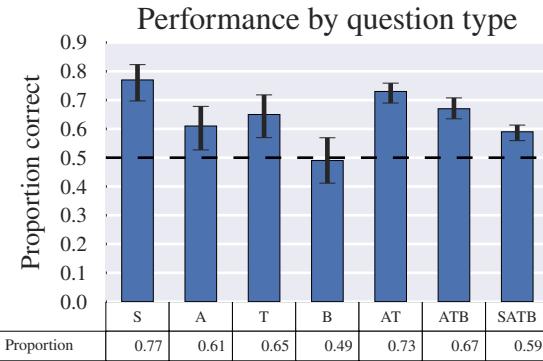


Fig. 6.6 responses-Mask

1 ?? shows the performance of BachBot on various question types. It shows that 59%
 2 fliang: VERIFY LAST

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

6 fliang: VERIFY LAST

7 better chance than randomly guessing when distinguishing Bach from BachBot cor-
 8 rectly.

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

12 fliang: reference

6.2 Results

81

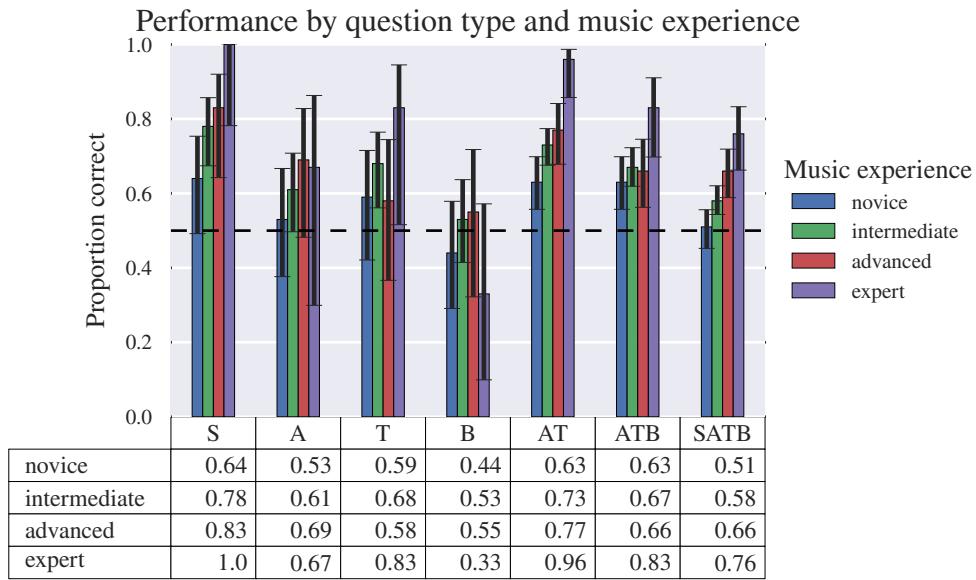


Fig. 6.7 responses-mask-MusicExperience

is uni-directional and does not account for any future constraints on other parts. We made this design decision intentionally because one of our requirements was sampling the model for novel compositions. However, since harmonization tasks provide the full past and future context for other parts, they effectively impose constraints on LSTM hidden state dynamics. We expect methods which account for both future and past context (e.g. using the output sequence from a bidirectional RNN

fliang: cite

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

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

fliang: cite

. Composing the alto and tenor parts achieved similar performance as composing all four parts, a result which may also be caused by not accounting for future constraints on model outputs. Removing the bass proved to be the most perceptually difficult to discern from real Bach.

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

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

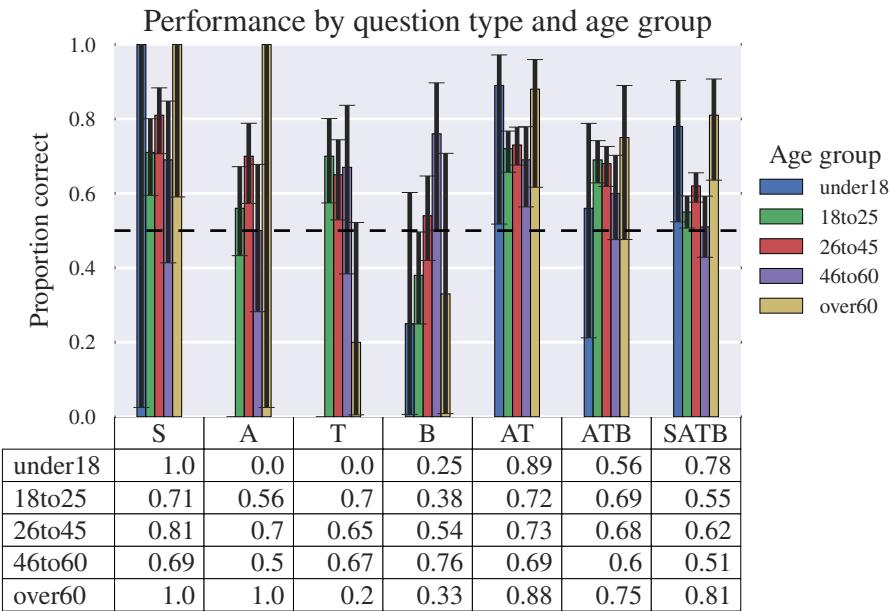


Fig. 6.8 responses-mask-Agegroup

fliang: VERIFY LAST

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

fliang: Have Mark analyze bad examples in ??

6.3 User feedback

The modulations and part writing were the giveaway for me (and once or twice the phrasing)
 Got 5/5. The trick is to listen for the unnatural pauses at regular intervals.
 Cool project, I scored 100% so I'm quite pleased with myself ;o) I do play an instrument
 although I'm not classical trained. If I had an inkling to why I could choose the background
 phrasing of the Bach pieces is far more elegant than the computer generated pieces.
 @samim @feynmanliang really impressive! If I didn't know about counterpoint that quiz
 would've stumped me

6.4 Competitive analysis of large-scale evaluation methodologies

83

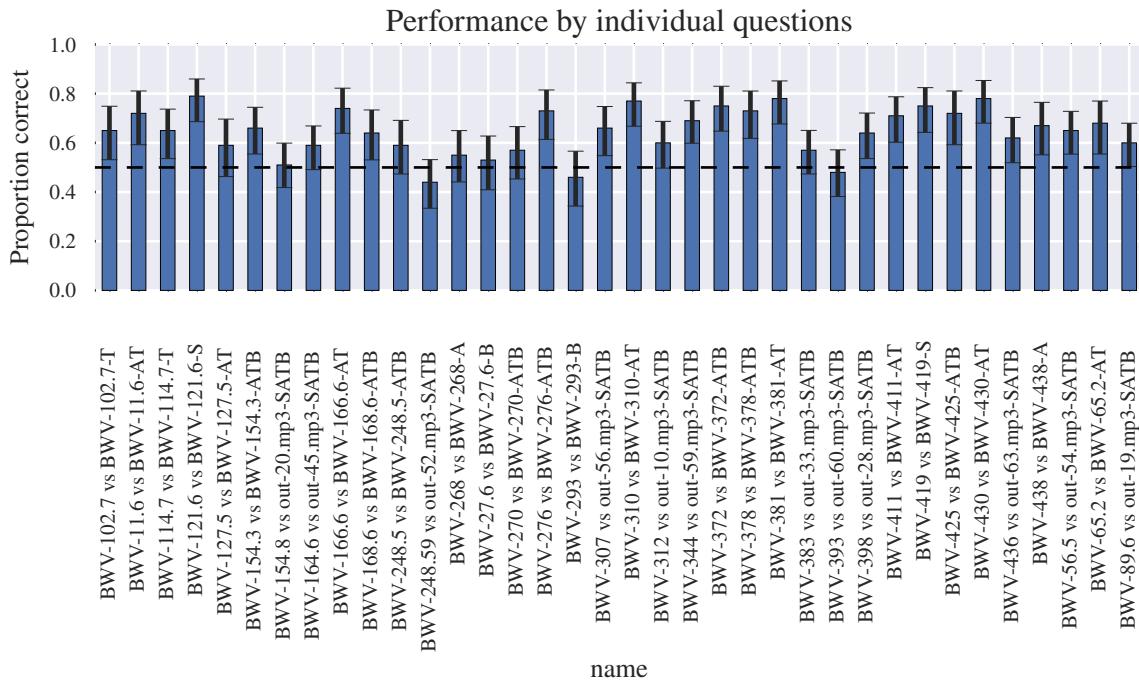


Fig. 6.9 Proportion of correct responses broken down by individual questions.

6.4 Competitive analysis of large-scale evaluation methodologies

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

fliang: Compare costs and quality with MTurk

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

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

¹ Paid service providers cost anywhere from \$20 to \$55 per month just for authoring tools and
² server space[107] At the time of writing, paid responses cost \$1.50–\$3.00 on SurveyMonkey
³ [uks].

7

Analysis of musical concepts learned by the model

7.1 Investigation of neuron activation responses to applied stimulus

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

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

fliang: cite preprocessing

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

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

fliang: ref

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

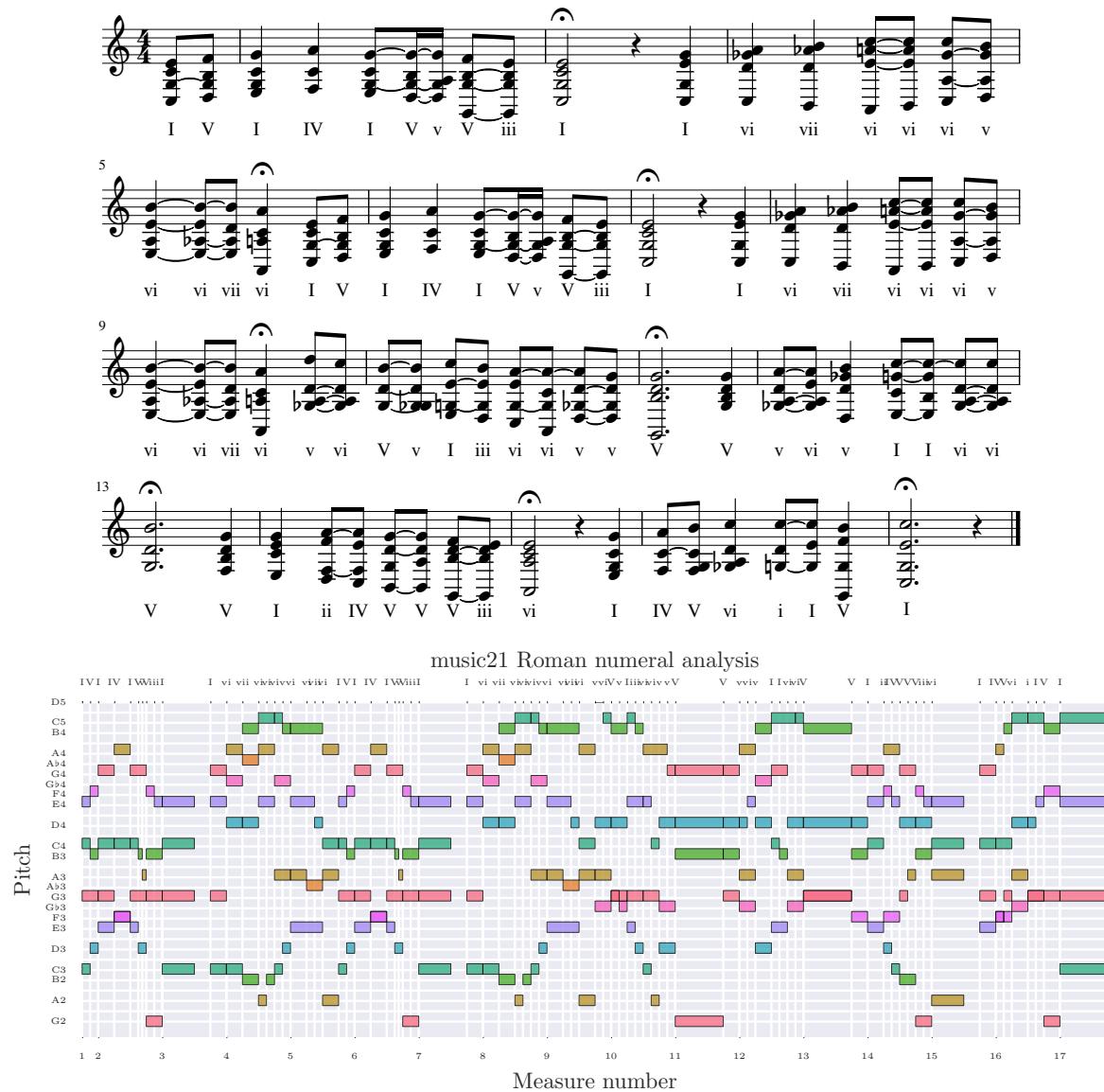
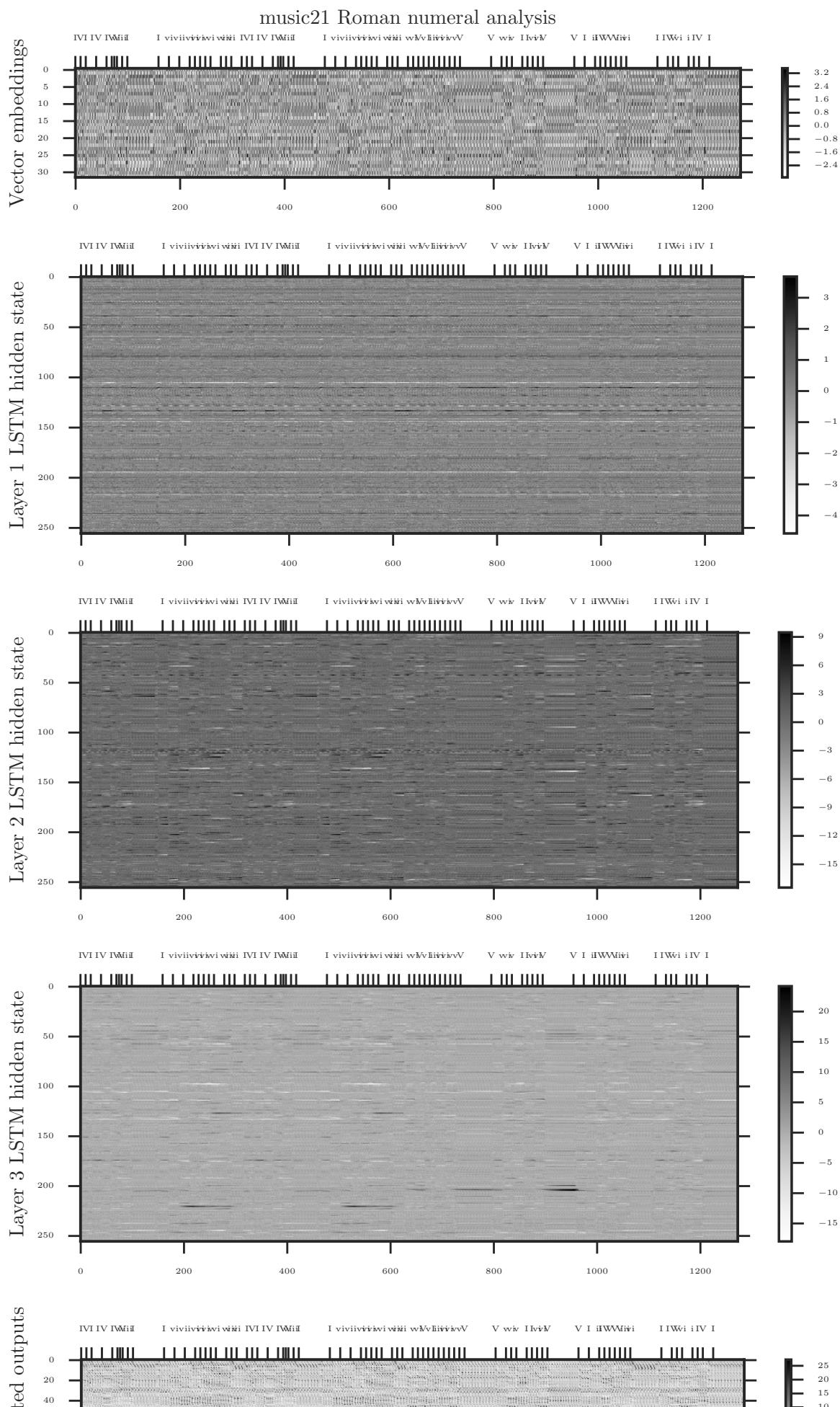


Fig. 7.1 *Top*: The preprocessed score (BWV 133.6) used as input stimulus with Roman numeral analysis annotations obtained from music21; *Bottom*: The same stimulus represented on a piano roll

7.1 Investigation of neuron activation responses to applied stimulus



7.1.1 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 (i.e. note-by-note) to lower resolution (i.e. frame-by-frame) is reminiscent of pooling operations in convolutional neural networks

fliang: cite

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

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

$$\tilde{\mathbf{z}}_n^{(l)} := \left[\max_{t_m < t < t_n} \mathbf{z}_{t,1}^{(l)}, \quad \max_{t_m < t < t_n} \mathbf{z}_{t,2}^{(l)}, \quad \dots, \quad \max_{t_m < t < t_n} \mathbf{z}_{t,N^{(l)}}^{(l)} \right]^T \quad (7.1)$$

where $\mathbf{z}_{t,i}^{(l)}$ 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 $\tilde{\mathbf{z}}$ is now indexed by frames rather than by tokens and hence corresponds to time-steps.

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

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

7.1.2 Probabilistic piano roll: likely variations of the stimulus

The bottom panel in ?? shows the model’s predictions for tokens in the next frames, where the tokens are arranged according to (arbitrary) index within the vocabulary. As the tokens correspond to pitches, they can be sorted according to pitch to reconstruct a **probabilistic piano roll**[28] consisting of the model’s sequence of next-frame predictions as it processes the input.

7.1 Investigation of neuron activation responses to applied stimulus

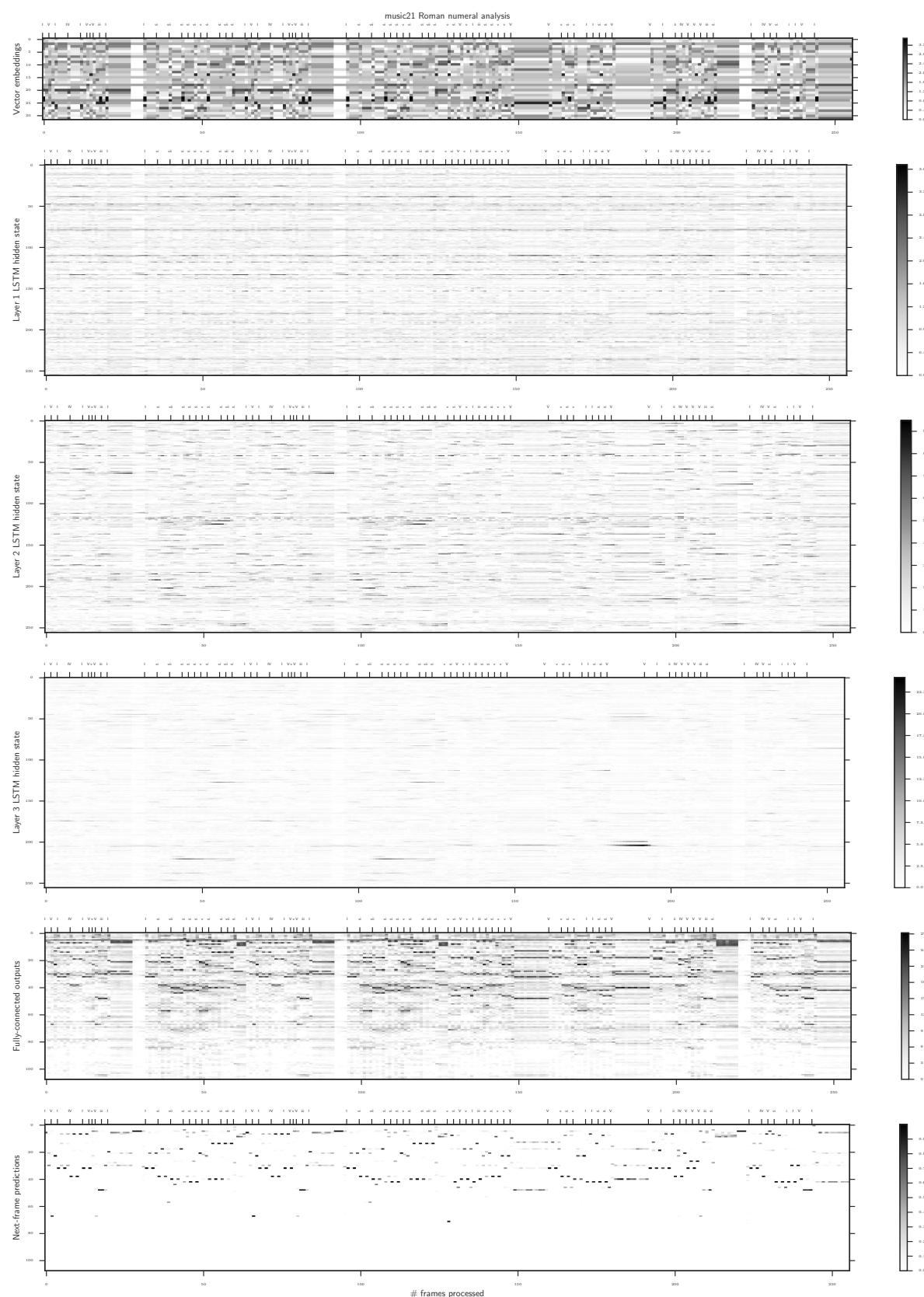


Fig. 7.3 Neuron activations over time pooled over frames

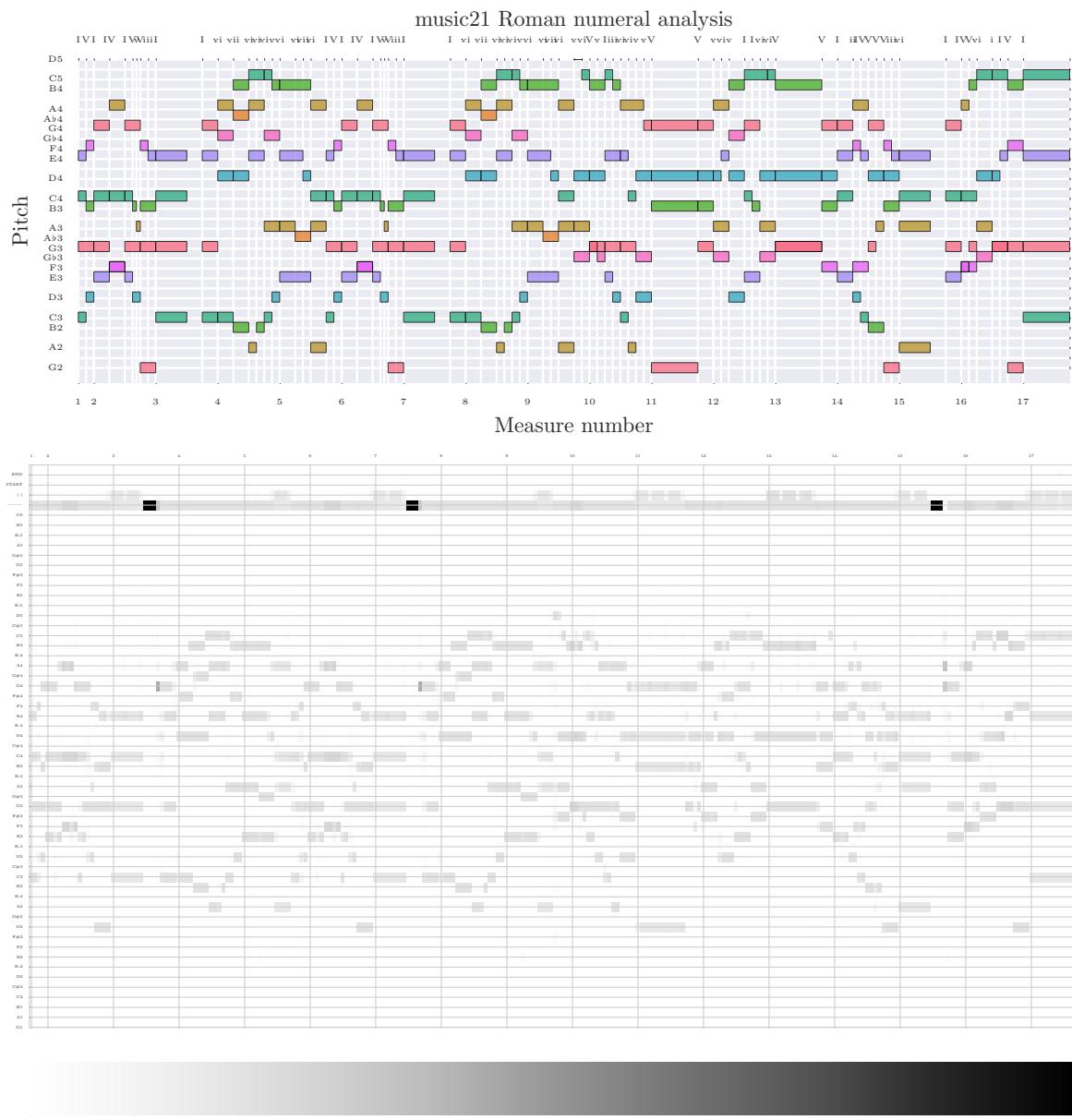


Fig. 7.4 Top: piano roll of stimulus (included for reference); Bottom: probabilistic piano roll

Notice that the probabilistic piano roll in ?? closely resembles the stimulus. This is unsurprising because the recurrent inputs are taken from the stimulus rather than sampled from the model’s predictions (a.k.a. [104]), so a model which predicts to only continue holding its input would produce a probabilistic piano roll identical to the stimulus delayed by one frame.

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

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

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

7.1.3 Neurons specific to musical concepts

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

fliang: Cite deconvolution

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

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

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

The first three neurons ((1, 64), (1, 138), (1, 207)) shown in the 2nd to 4th panel from top of ?? effectively behave like cadence detectors. While they all exhibit activity when the stimulus contains V chords (i.e. G-major). (1, 64) and (1, 138) are both specific to perfect cadences (i.e. $V - I$ chord progressions) used to conclude phrases and differ only in the chord inversions

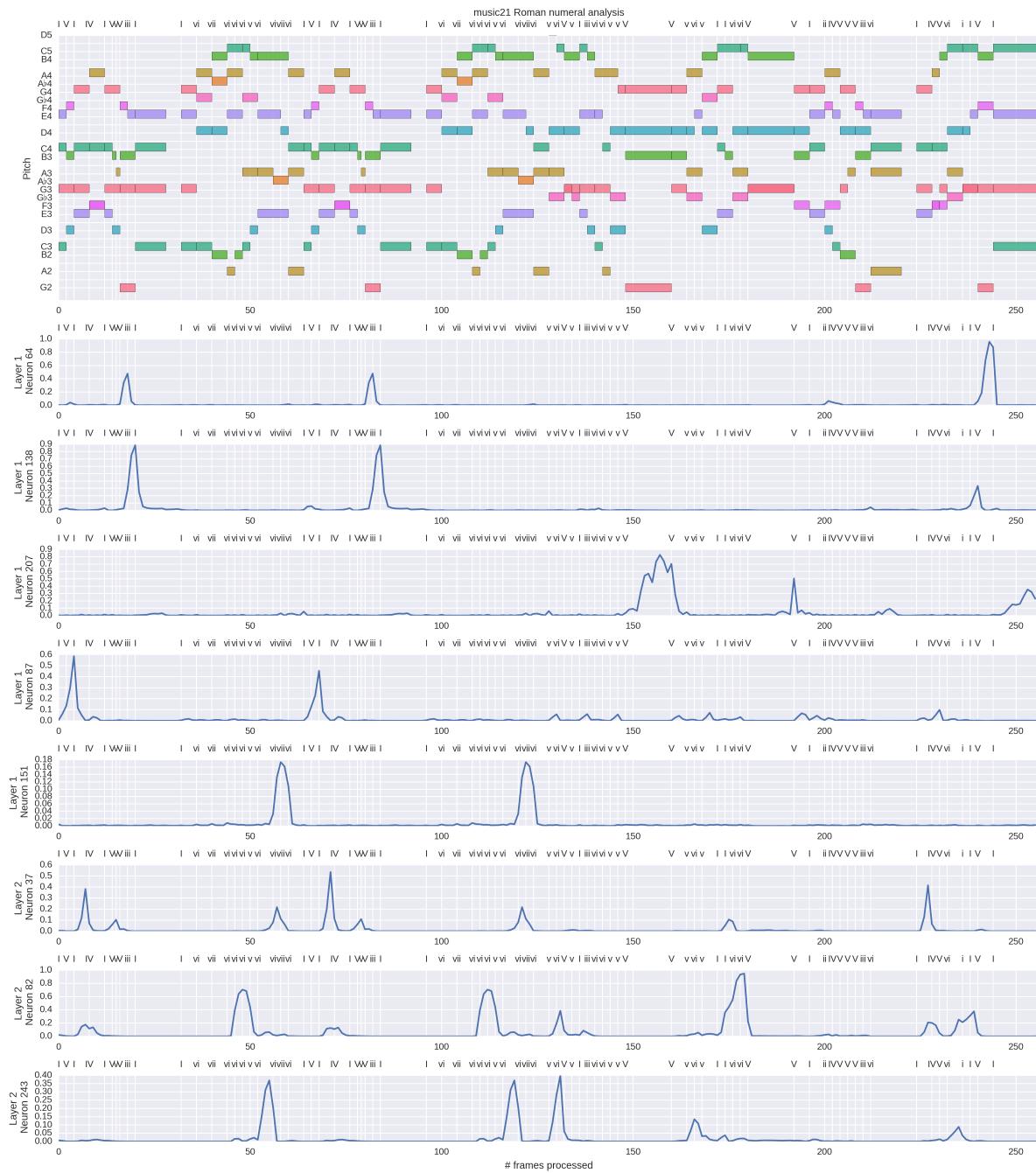


Fig. 7.5 Activation profiles of neurons within our model which have learned high-level musical concepts

which they are most sensitive to. In contrast, (1, 207) only exhibits activity for the V chord associated with the imperfect cadences near frames 150 and 180.

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

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

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

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

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

8

1

Summary and Conclusions

2

8.1 Contributions

3

8.2 Findings and conclusions

4

8.3 Future work

5

Bidirectional RNNs

6

N-best list instead of 1-best search, RNN lattice rescoring (@Marcin)

7

Different ordering of SATB when encoding

8

Bigger data sets, larger degrees of polyphony, different musical styles

9

Listening test, subjective free-text feedback

10

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

References

- [uks] Online research panel pricing | surveymonkey audience. 2
- [2] (2015). abc:standard [abc wiki]. 3
- [3] Allan, M. and Williams, C. K. (2005). allan2005. *Advances in Neural Information Processing Systems*, 17:25–32. 4
- [4] Ariza, C. (2009). The interrogator as critic: The turing test and the evaluation of generative music systems. *Computer Music Journal*, 33(2):48–70. 5
- [5] Ariza, C. and Cuthbert, M. (2010). *Modeling beats, accents, beams, and time signatures hierarchically with music21 meter objects*. Ann Arbor, MI: Michigan Publishing, University of Michigan Library. 6
- [6] Bellgard, M. I. and Tsang, C.-P. (1994). Harmonizing music the boltzmann way. *Connection Science*, 6(2-3):281–297. 7
- [7] Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179. 8
- [8] Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127. 9
- [9] Bengio, Y. and Delalleau, O. (2011). On the expressive power of deep architectures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6925 LNAI:18–36. 10
- [10] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166. 11
- [11] Bharucha, J. J. and Todd, P. M. (1989). Modeling the perception of tonal structure with neural nets. *Computer Music Journal*, 13(4):44–53. 12
- [12] Boulanger-Lewandowski, N., Vincent, P., and Bengio, Y. (2012). Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription. *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, (Cd):1159–1166. 13
- [13] Brien, T. O. and Roman, I. (2016). A Recurrent Neural Network for Musical Structure Processing and Expectation. *CS224d: Deep Learning for Natural Language Processing Final Projects*, pages 1–9. 14

- ¹ [14] Butt, J. (1999). Bach-werke-verzeichnis. *Notes*, 55(4):890–893.
- ² [15] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.
- ⁵ [16] Cooper, G. and Meyer, L. B. (1963). *The rhythmic structure of music*, volume 118. University of Chicago Press.
- ⁷ [17] Cope, D. (1992). Computer modeling of musical intelligence in emi. *Computer Music Journal*, 16(2):69–83.
- ⁹ [18] Coutinho, E., Gimenes, M., Martins, J. M., and Miranda, E. R. (2005). Computational musicology: An artificial life approach. In *2005 portuguese conference on artificial intelligence*, pages 85–93. IEEE.
- ¹² [19] Cruz-Alcázar, P. P. and Vidal-Ruiz, E. (1998). Learning regular grammars to model musical style: Comparing different coding schemes. In *International Colloquium on Grammatical Inference*, pages 211–222. Springer.
- ¹⁵ [20] Cuthbert, M. S. and Ariza, C. (2010). music21: A toolkit for computer-aided musicology and symbolic music data.
- ¹⁷ [21] Cuthbert, M. S., Cabal-ugaz, J., Ariza, C., and Hadley, B. (2011). Hidden Beyond MIDI’s Reach : Feature Extraction and Machine Learning with Rich Symbolic Formats in music21. *Proceedings of the Neural Information Processing Systems Conference*, pages 3–4.
- ²⁰ [22] Cybenko, G. (1993). Degree of approximation by superpositions of a sigmoidal function. *Approximation Theory and its Applications*, 9(3):17–28.
- ²² [23] Dannenberg, R. B., Thom, B., and Watson, D. (1997). A machine learning approach to musical style recognition.
- ²⁴ [24] Denny, J. (1960). *The Oxford school harmony course*, volume 1. Oxford University Press.
- ²⁵ [25] Denton, C. and Fillion, M. (1997). The history of musical tuning and temperament during the classical and romantic periods.
- ²⁷ [26] Downs, J. S., Holbrook, M. B., Sheng, S., and Cranor, L. F. (2010). Are your participants gaming the system?: screening mechanical turk workers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2399–2402. ACM.
- ³⁰ [27] Ebcioğlu, K. (1988). An expert system for harmonizing four-part chorales. *Computer Music Journal*, 12(3):43–51.
- ³² [28] Eck, D. and Lapalme, J. (2008). Learning musical structure directly from sequences of music. *University of Montreal, Department of Computer Science, CP*, 6128.
- ³⁴ [29] Eck, D. and Schmidhuber, J. (2002a). A First Look at Music Composition using LSTM Recurrent Neural Networks. *Idsia*.

-
- [30] Eck, D. and Schmidhuber, J. (2002b). Finding temporal structure in music: Blues improvisation with LSTM recurrent networks. *Neural Networks for Signal Processing - Proceedings of the IEEE Workshop*, 2002-Janua:747–756. 1
2
3
- [31] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211. 4
- [32] Feulner, J. and Hörnel, D. (1994). Melonet: Neural networks that learn harmony-based melodic variations. In *Proceedings of the International Computer Music Conference*, pages 121–121. INTERNATIONAL COMPUTER MUSIC ACCOCIATION. 5
6
7
- [33] Franklin, J., Dietterich, T., Becker, S., and Ghahramani, Z. (2001). Learning and improvisation. In *Neural Information Processing Systems*, volume 14. 8
9
- [34] Franklin, J. A. (2004a). Predicting reinforcement of pitch sequences via lstm and td. In *Proc. of International Computer Music Conference, Miami, Florida*, volume 306. 10
11
- [35] Franklin, J. A. (2004b). Recurrent neural networks and pitch representations for music tasks. In *FLAIRS Conference*, pages 33–37. 12
13
- [36] Franklin, J. A. (2005). Jazz melody generation from recurrent network learning of several human melodies. In *FLAIRS Conference*, pages 57–62. 14
15
- [37] Franklin, J. A. (2006). Recurrent neural networks for music computation. *INFORMS Journal on Computing*, 18(3):321–338. 16
17
- [38] Freedman, D. (2015). *Correlational Harmonic Metrics: Bridging Computational and Human Notions of Musical Harmony*. PhD thesis. 18
19
- [39] Gers, F. A., Pérez-Ortiz, J. A., Eck, D., and Schmidhuber, J. (2002a). Dekf-lstm. In *ESANN*, pages 369–376. 20
21
- [40] Gers, F. A. and Schmidhuber, E. (2001). Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340. 22
23
24
- [41] Gers, F. A. and Schmidhuber, J. (2000). Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE. 25
26
27
- [42] Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471. 28
29
- [43] Gers, F. A., Schraudolph, N. N., and Schmidhuber, J. (2002b). Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143. 30
31
- [44] Goller, C. and Kuchler, A. (1996). Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pages 347–352. IEEE. 32
33
34
- [45] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680. 35
36
37

- 1 [46] Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bi-
2 directional LSTM networks. *Proceedings of the International Joint Conference on Neural
3 Networks*, 4:2047–2052.
- 4 [47] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2015).
5 Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*.
- 6 [48] Griffith, N. and Todd, P. M. (1999). *Musical networks: Parallel distributed perception
7 and performance*. MIT Press.
- 8 [49] Handel, S. (1993). *Listening: An introduction to the perception of auditory events*. The
9 MIT Press.
- 10 [50] Heafield, K., Pouzyrevsky, I., Clark, J. H., and Koehn, P. (2013). Scalable modified
11 Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the
12 Association for Computational Linguistics*, Sofia, Bulgaria.
- 13 [51] Herlands, W., Der, R., Greenberg, Y., and Levin, S. (2014). A Machine Learning Ap-
14 proach to Musically Meaningful Homogeneous Style Classification. *Twenty-Eighth AAAI
15 Conference on Artificial Intelligence*, pages 276–282.
- 16 [52] Hild, H., Feulner, J., and Menzel, W. (1991). Harmonet: A neural net for harmonizing
17 chorales in the style of js bach. In *NIPS*, pages 267–274.
- 18 [53] Hinton, G. E. and Sejnowski, T. J. (1986). Learning and relearning in boltzmann ma-
19 chines. *Parallel distributed processing: Explorations in the microstructure of cognition*,
20 1:282–317.
- 21 [54] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computa-
22 tion*, 9(8):1735–1780.
- 23 [55] Hörmel, D. (1997). Melonet i: Neural nets for inventing baroque-style chorale variations.
24 In *NIPS*, pages 887–893.
- 25 [56] Hornel, D. and Ragg, T. (1996). Learning musical structure and style by recognition,
26 prediction and evolution.
- 27 [57] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network train-
28 ing by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.
- 29 [58] Jordan, M. I. (1997). Serial order: A parallel distributed processing approach. *Advances
30 in psychology*, 121:471–495.
- 31 [59] Koutník, J., Greff, K., Gomez, F., and Schmidhuber, J. (2014). A Clockwork RNN.
32 *Proceedings of The 31st International Conference on Machine Learning*, 32:1863–1871.
- 33 [60] Krumhansl, C. L. (2001). *Cognitive foundations of musical pitch*. Oxford University
34 Press.
- 35 [61] Laden, B. and Keefe, D. H. (1989). The representation of pitch in a neural net model of
36 chord classification. *Computer Music Journal*, 13(4):12–26.

-
- [62] Lerdahl, F. (1983). Jackendoff. a generative theory of tonal music. 1
- [63] Linnainmaa, S. (1970). The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, pages 6–7. 2
3
4
- [64] Liu, I.-T. and Ramakrishnan, B. (2014). Bach in 2014: Music Composition with Recurrent Neural Network. *arXiv:1412.3191*, 5:1–9. 5
6
- [65] Liu, X., Wang, Y., Chen, X., Gales, M. J., and Woodland, P. C. (2014). Efficient lattice rescoring using recurrent neural network language models. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4908–4912. IEEE. 7
8
9
- [66] Lyu, Q. (2015). Polyphonic Music Modelling with LSTM-RTRBM. *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pages 991–994. 10
11
- [67] Mandel, M. I., Poliner, G. E., and Ellis, D. P. (2006). Support vector machine active learning for music retrieval. *Multimedia systems*, 12(1):3–13. 12
13
- [68] Mikolov, T. (2012). *Statistical Language Models Based on Neural Networks*. PhD thesis. 14
- [69] Mikolov, T., Joulin, A., Chopra, S., and Mathieu, M. (2015). Learning Longer Memory in Recurrent Neural Networks. *Iclr*, pages 1–9. 15
16
- [70] Mikolov, T., Karafiat, M., Burget, L., Cernocky, J., and Khudanpur, S. (2010). Recurrent Neural Network based Language Model. *Interspeech*, (September):1045–1048. 17
18
- [71] Mozer, M. C. (1994). Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2–3):247–280. 19
20
21
- [72] Nattiez, J.-J. (1990). *Music and discourse: Toward a semiology of music*. Princeton University Press. 22
23
- [73] Nayebi, A. and Vitelli, M. (2015). GRUV: Algorithmic Music Generation using Recurrent Neural Networks. *CS224d: Deep Learning for Natural Language Processing Final Projects*, pages 1–6. 24
25
26
- [74] Ni, Y., McVicar, M., Santos-Rodriguez, R., and De Bie, T. (2012). An end-to-end machine learning system for harmonic analysis of music. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(6):1771–1783. 27
28
29
- [75] Oswald, J. (1973). Harmony: Schenkerian analysis. 30
- [76] Papadopoulos, G. and Wiggins, G. (1999). Ai methods for algorithmic composition: A survey, a critical view and future prospects. In *AISB Symposium on Musical Creativity*, pages 110–117. Edinburgh, UK. 31
32
33
- [77] Pascanu, R., Mikolov, T., and Bengio, Y. (2012). On the difficulty of training recurrent neural networks. *Proceedings of The 30th International Conference on Machine Learning*, (2):1310–1318. 34
35
36

- ¹ [78] Pearce, M., Meredith, D., and Wiggins, G. (2002). Motivations and methodologies for
² automation of the compositional process. *Musicae Scientiae*, 6(2):119–147.
- ³ [79] Pearce, M. and Wiggins, G. (2001). Towards a framework for the evaluation of machine
⁴ compositions. In *Proceedings of the AISB'01 Symposium on Artificial Intelligence and*
⁵ *Creativity in the Arts and Sciences*, pages 22–32. Citeseer.
- ⁶ [80] Piston, W. (1978). Harmony. (revised and expanded by mark devoto). *Londres: Victor*
⁷ *Gollancz LTD*.
- ⁸ [81] Ramage, D. (2007). Hidden markov models fundamentals. *Lecture Notes*. <http://cs229.stanford.edu/section/cs229-hmm.pdf>.
- ¹⁰ [82] Randel, D. M. (1999). *The Harvard concise dictionary of music and musicians*. Harvard
¹¹ University Press.
- ¹² [83] Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation
¹³ learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference*
¹⁴ *On*, pages 586–591. IEEE.
- ¹⁵ [84] Robinson, A. and Fallside, F. (1987). *The utility driven dynamic error propagation net-*
¹⁶ *work*. University of Cambridge Department of Engineering.
- ¹⁷ [85] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by
¹⁸ back-propagating errors. *Cognitive modeling*, 5(3):1.
- ¹⁹ [86] Shepard, R. N. (1982). Geometrical approximations to the structure of musical pitch.
²⁰ *Psychological review*, 89(4):305.
- ²¹ [87] Spangler, R. R., Goodman, R. M., and Hawkins, J. (1998). Bach in a box-real-time
²² harmony.
- ²³ [88] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014).
²⁴ Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine*
²⁵ *Learning Research*, 15(1):1929–1958.
- ²⁶ [89] Stamatatos, E. and Widmer, G. (2005). Automatic identification of music performers
²⁷ with learning ensembles. *Artificial Intelligence*, 165(1):37–56.
- ²⁸ [90] Stolcke, A. et al. (2002). Srilm-an extensible language modeling toolkit. In *Interspeech*,
²⁹ volume 2002, page 2002.
- ³⁰ [91] Sturm, B., Santos, J. F., and Korshunova, I. (2015). Folk music style modelling by recur-
³¹ rent neural networks with long short term memory units. In *16th International Society for*
³² *Music Information Retrieval Conference*.
- ³³ [92] Sturm, B. L., Santos, J. F., Ben-Tal, O., and Korshunova, I. (2016). Music transcription
³⁴ modelling and composition using deep learning. *arXiv preprint arXiv:1604.08723*.
- ³⁵ [Sutskever] Sutskever, I. Training Recurrent Neural Networks - Ilia Sutskever - PhD thesis.
- ³⁶ [94] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, vol-
³⁷ ume 1. MIT press Cambridge.

- [95] Terhardt, E. (1974). Pitch, consonance, and harmony. *The Journal of the Acoustical Society of America*, 55(5):1061–1069. 1
2
- [96] Todd, P. (1988). A sequential network design for musical applications. In *Proceedings of the 1988 connectionist models summer school*, pages 76–84. 3
4
- [97] Todd, P. M. (1989). A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43. 5
6
- [98] Toivainen, P. (2000). Symbolic ai versus connectionism in music research. 7
- [99] Tsang, C. P. and Aitken, M. (1991). Harmonizing music as a discipline in constraint logic programming. In *Proceedings of the International Computer Music Conference*, pages 61–61. INTERNATIONAL COMPUTER MUSIC ACCOCIATION. 8
9
10
- [100] Tymoczko, D. (2009). Three conceptions of musical distance. *Communications in Computer and Information Science*, 38:258–272. 11
12
- [101] Wanner, E. (1980). The atn and the sausage machine: Which one is baloney? *Cognition*, 8(2):209–225. 13
14
- [102] White, J. D. and Lake, W. E. (2002). *Guidelines for college teaching of music theory*. Scarecrow Press. 15
16
- [103] Williams, R. J. and Peng, J. (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501. 17
18
- [104] Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280. 19
20
- [105] Williams, R. J. and Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. *Back-propagation: Theory, architectures and applications*, pages 433–486. 21
22
23
- [106] Winograd, T. (1968). Linguistics and the computer analysis of tonal harmony. *Journal of Music Theory*, 12(1):2–49. 24
25
- [107] Wright, K. B. (2005). Researching internet-based populations: Advantages and disadvantages of online survey research, online questionnaire authoring software packages, and web survey services. *Journal of Computer-Mediated Communication*, 10(3):00–00. 26
27
28
- [108] Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*. 29
30

¹ Fig. 1 Results of grid search (see ??) over LSTM sequence model hyperparameters

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

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
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
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

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
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
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

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
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
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

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
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
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

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
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
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

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

A

Installing the CUED class file

$\text{\LaTeX}.\text{cls}$ files can be accessed system-wide when they are placed in the $\langle\text{texmf}\rangle/\text{tex}/\text{latex}$ directory, where $\langle\text{texmf}\rangle$ is the root directory of the user's \TeX installation. On systems that have a local texmf tree ($\langle\text{texmflocal}\rangle$), which may be named “texmf-local” or “localtexmf”, it may be advisable to install packages in $\langle\text{texmflocal}\rangle$, rather than $\langle\text{texmf}\rangle$ as the contents of the former, unlike that of the latter, are preserved after the \LaTeX system is reinstalled and/or upgraded.

It is recommended that the user create a subdirectory $\langle\text{texmf}\rangle/\text{tex}/\text{latex}/\text{CUED}$ for all CUED related \LaTeX class and package files. On some \LaTeX systems, the directory look-up tables will need to be refreshed after making additions or deletions to the system files. For $\text{\TeX} \text{Live}$ systems this is accomplished via executing “texhash” as root. MIK \TeX users can run “initexmf -u” to accomplish the same thing.

Users not willing or able to install the files system-wide can install them in their personal directories, but will then have to provide the path (full or relative) in addition to the filename when referring to them in \LaTeX .

Draft - v1.0

Tuesday 9th August, 2016 – 05:02

9

Graveyard

9.1 Neural Networks

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

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

fliang: CITE: fig:nn-layer

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

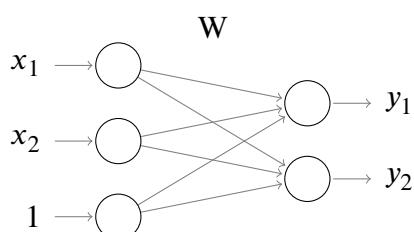


Fig. 9.1 Single feedforward neural network layer

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

1 fliang: CITE: fig:ffw-nn

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

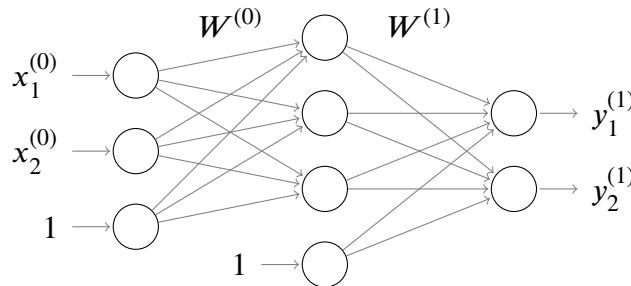


Fig. 9.2 2-layer feedforward neural network

4 When discussing neural networks with $L \geq 1$ layers, we will use $\mathbf{x}^{(i)}$, $\mathbf{W}^{(i)}$, $\mathbf{z}^{(i)}$, and $\mathbf{y}^{(i)}$ to
 5 refer to the inputs, weights, activations, and outputs of the i th layer. The activation function σ
 6 is understood to act elementwise when applied to a vector. For adjacent layers $i, i + 1$, we have
 7 $\mathbf{x}^{(i+1)} = \mathbf{y}^{(i)}$. $\mathbf{x}^{(0)}$ and $\mathbf{y}^{(L)}$ are the inputs and outputs respectively of the entire network.

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

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

11 Modeling probability distributions

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

15 fliang: CITE: eq:softmax

16 .

$$\small 17 P(o = k | \mathbf{z}^{(L)}) = \frac{\exp -z_k^{(L)}}{\sum_j \exp -z_j^{(L)}} \quad (9.1)$$

18 Efficient gradient computations through back-propagation

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

fliang: cite backprop

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

fliang: cite theano

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

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

fliang: Cite

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

Vanishing gradients

The solution is to rewrite

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

such that

fliang: CITE: eq:prod-hi

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

$$h_t = h_{t-1} + \theta_x x_t \quad (9.2)$$

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

Training with back-propogation

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

fliang: CITE: eq:cross-entropy-loss

):

$$\begin{aligned}
 1 \quad L(\mathbf{z}_{1:N}^{(L)}, \hat{o}_{1:N}) &= \sum_{i=1}^N L(\mathbf{z}_i^{(L)}, \hat{o}_i) && \text{Independence across samples} \\
 2 \quad &= \sum_i \sum_k \delta_{\hat{o}_i, k} \log \frac{1}{P(o=k|\mathbf{y}_i^{(L)})} \\
 3
 \end{aligned} \tag{9.3}$$

4 Gradient descent proceeds by using the Jacobian (i.e. gradient) $\nabla_{\theta} L(\mathbf{z}_{1:N}^{(L)}, \hat{o}_{1:N})$ to iteratively update the network parameters using successive first-order approximations (\\
 5 fliang: CITE: eq:nn-training-iteration-scheme \\
 6). \\
 7

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

10 Variants of \\
 11 fliang: CITE: eq:nn-training-iteration-scheme

12 which adaptively set the step size η_t or incorporate/estimate the Hessian $\nabla_{\theta}^2 L(\cdot, \cdot)$ can yield \\
 13 performance when applied to neural network training. However, their discussion is out of \\
 14 scope. \\
 15 fliang: Discuss RMSprop?

16 To apply \\
 17 fliang: CITE: eq:nn-training-iteration-scheme

18 , the gradient $\nabla_{\theta} L(\mathbf{z}_{1:N}^{(L)}, \hat{o}_{1:N})$ must be computed. This can be accomplished using **back-** \\
 19 **propagation** \\
 20 fliang: cite

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

22 Let $\delta_j^{(l)} = \frac{\partial L(\mathbf{z}_{1:N}^{(L)}, \hat{o}_{1:N})}{\partial z_j^{(l)}}$ be the partial derivative of the loss with respect to the j th activation \\
 23 of layer l . For the final L th layer, cross-entropy loss with a Boltzmann distribution yields \\
 24 fliang: CITE: eq:cross-entropy-loss

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

For earlier layers $l < L$, we have

$$\delta_j^{(l)} = \sum_k \frac{\partial L(\mathbf{z}_{1:N}^{(L)}, \hat{o}_{1:N})}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} \quad (9.5)$$

$$= \sum_k \delta_k^{(l+1)} \frac{\partial}{\partial z_j^{(l)}} (\mathbf{W}^{(l+1)}[\sigma(z^{(l)}), 1]^\top)_k \quad (9.6)$$

$$= \sum_k \delta_k^{(l+1)} \mathbf{W}_{k,j}^{(l+1)} \sigma'(z_j^{(l)}) \quad (9.7)$$

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

fliang: DO THIS

$$\circ \quad (9.8)$$

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

The back-propagation algorithm consists of two steps:

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

2. *Backward pass:* Recursively iterate

fliang: CITE: eq:backprop

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

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

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

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

Some appealing properties of backpropogation:

- Efficient exploitation of the computation graph: chain rule expansions are constrained by the computation graph, improving efficiency because factors which don't contribute to a given $\delta^{(l)}$ are neglected in the recursion
- Implementation using local rules: the forward/backward pass at any layer l only requires knowledge of $z^{(l)}$, $\delta^{(l+1)}$, and the derivative of the activation σ' . As all these quantities are localized to one layer, this permits modular implementations where a node which can be back-propogated through needs only implement a `forward()` method which computes activations given inputs and a `backward()` method which computes $\delta^{(l)}$ given activations.

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

9.2 RNNs

The advantages of RNNs over feedforward networks include:

- Ability to handle variable-length inputs: the RNN can be unrolled an arbitrary number of times to accomodate inputs x of different length
- Fixed dimension embeddings: after processing the entirety of an input sequence, the state of the RNN can be used as a fixed dimension embedding representing the input
- Sequential processing: the order of $x_{1:T}$ will affect the state trajectory $s_{1:T}$, enabling the model to capture time-dependent dynamics within the input sequences
- Memory over time: the state $s \in \mathbb{R}^D$ can take on an uncountably infinite number of values, allowing it to potentially act as memory which summarizes *all* of the input up to the current time

Comparison against HMMs

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

fliang: Define HMMs

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

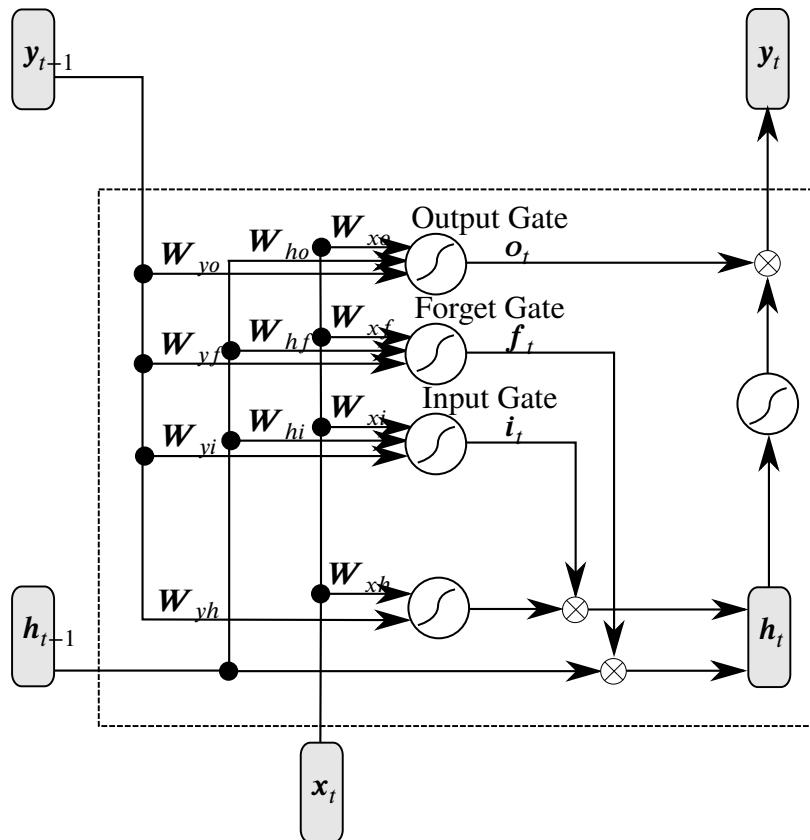


Fig. 9.3 Single LSTM unit

9.3 Sequence probability modelling

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

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

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

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

6 **9.4 Related work**

7 [21] used music21 to generate rich feature representations for music for downstream machine
8 learning tasks.

9 The application of machine learning to music has a rich history. [51] describe a system to
10 classify music into homogeneous styles. However, they focus on the discriminative task and
11 do not consider how to generate novel scores.

12 **9.5 LSTMs: background and motivation**

13 Two prominent methods for training RNNs include real-time recurrent learning (RTRL) [84]
14 and backpropagation through time (BPTT) [105]. [103] introduces truncated BPTT to address
15 computational complexity when learning over very long sequences. Temporal difference [94]
16 has also been proposed as a method for learning RNNs [34].

17 The first LSTM models, which did not include forget gates, was introduced in [54]. [42]
18 later revised the LSTM model to include forget gates in order to prevent hidden states from
19 growing indefinitely.

20 LSTMs have been demonstrated to outperform traditional RNNs on a variety of tasks. [40]
21 demonstrates a LSTM correctly recognizing 1000 instances from the context-free grammar
22 $A^n B^n$ while an Elman RNN achieves only 20% accuracy.

23 Online adaptation at test time using a Kalman filter was described in [39]. [70] [68] refers
24 to this as “dynamic evaluation.”

25 In *Bach in a Box* [87], harmonic rules are collected in a database and then used to build
26 rule-based neural networks. This enables encoding of prior knowledge as rules in the rulebase.

27 [28] attempts to model meter by introducing time-delayed connections in [29]

28 **9.5.1 Representation of music data**

29 [71] discusses the importance of music representation, settling on *psychologically-based rep-*
30 *resentations* of pitch, duration, and harmonic structure [86].

Many attempts to represent musical data have been investigated. Attempts which explicitly model harmonic structure include a Circle of Thirds representation [35] or overlapping subharmonics representation[61], both of which have been studied in the context of generative RNN models [35] [71]. Other representations attempt to model notions such as musical distance in terms of voice leading, orbifolds, and tuning lattices[100].

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

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

9.5.2 Evaluation of models

[79] addresses difficulty in quantitative evaluation, suggesting the use of a learned critic in a manner similar to GANs [45]. In a later report, [78] attribute difficulty in evaluation due to lack of aim: algorithmic composition, design of compositional tools, and computational modelling of musical styles or music cognition all have different motivations and should thus be evaluated differently.

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

9.6 Automatic Composition

9.6.1 Multi-GPU implementation

To accelerate model training, we parallelize models across multiple GPUs. This is possible thanks to the summation operation in noisy gradient estimators:

$$\frac{1}{N} \sum_{i=1}^N \nabla L_i(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla L_i(\theta) \quad (9.11)$$

fliang: Real citations on noisy gradient

In particular, training RNNs with hidden state requires sequential traversal of the dataset. Parallelizing sequential iteration is accomplished by first segmenting into equal length segments and then initializing parallel iterators each pointing at a different segment. Each iterator sequentially reads data into GPU memory.

Model parameters are broadcast out to all GPUs on each forward pass and gradients are accumulated during each backward pass.

1 Research in grid LSTMs suggests that we can go deeper by introducing gates along the
 2 depth dimension to help permit information flow

3 fliang: cite gird LSTMs

4 .

5 9.7 Token-level embeddings

6 fliang: EXPERIMENT: redo these

7 Filter to notes

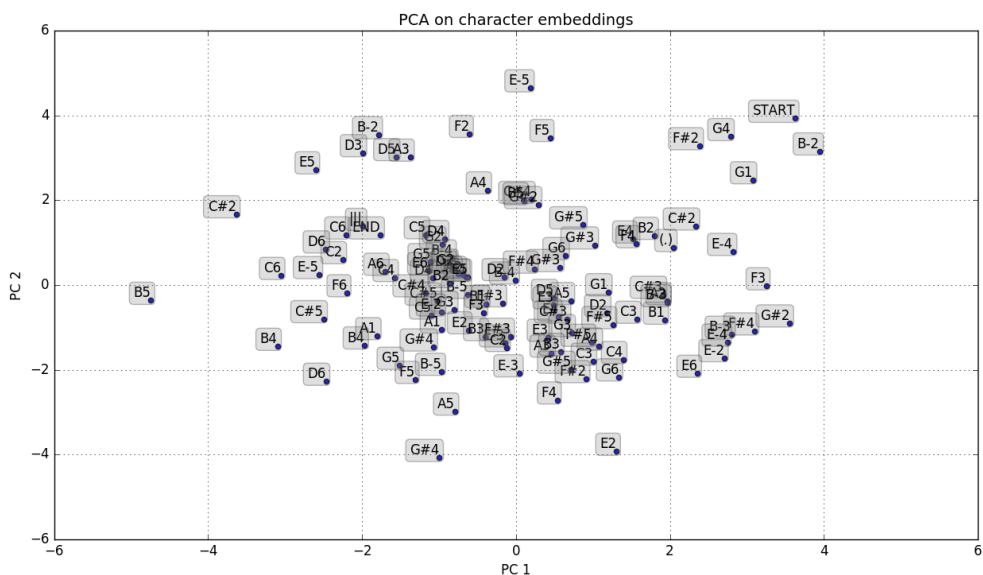


Fig. 9.4 PCA embedding of note tokens

8 9.7.1 Variable-length embeddings

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

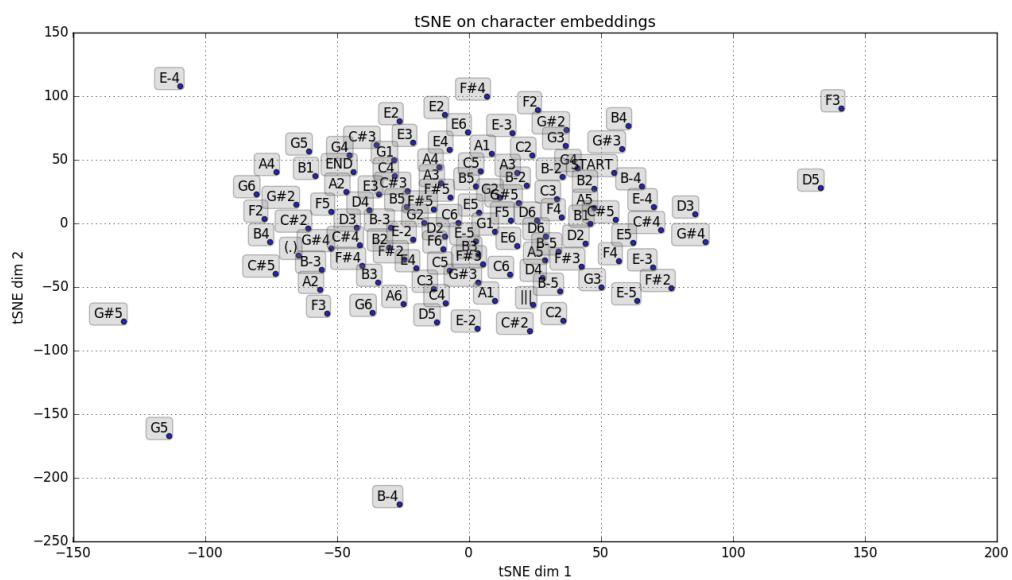
9.7 Token-level embeddings

Fig. 9.5 tSNE embedding of note tokens