Deep probabilistic sequence modelling of polyphonic music

An automatic composition system for Bach chorales

Feynman Liang

Department of Engineering
University of Cambridge

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

This dissertation is submitted for the degree of
Masters of Philosophy

Churchill College August 2016

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

Declaration

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

Total word count:

fliang: Fill this in at the end

Signed:

Date:

Feynman Liang
August 2016

Acknowledgements

And I would like to acknowledge ...

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

Marcin Tomczak : for ???

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

Abstract

This is where you write your abstract ...

Table of contents

List of figures

List of tables

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

Alpern [4]

1
Introduction

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.

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.

Our contributions incude:

1. A note-by-note sequential representation for polyphonic music amenable to processing with standard sequence models
2. A rigorous investigation of how recent deep learning advances such how as dropout [96], batch normalization [64], and new RNN architectures can be applied to improve probabilistic modelling of music data

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

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

   fliang: XXX

   participants

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

In this work, we investigate how sequence probability models parameterized by deep recurrent neural networks can be used as generative models over scores of music. Such a model has a variety of applications within computational music theory. The aim of this work is to investigate applications on two particular tasks: melody harmonization and automatic composition.

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

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

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

We will examine how design decisions made when constructing probability models over music affect the musical characteristics of generated samples, investigate practical matters encountered 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 varios 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.

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 [41]

.

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 [19].

fliang: Describe automatic composition vs harmonization vs analysis/scoring

Generative models are preferrable 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 [84] 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)

Pearce et al. [86] emphasizes *algorithmic composition* should "be applied exclusively to activities in which the motivations have this artistic character."

fliang: Describe piano roll vs sheet music

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

fliang: Motivate quantization with Western music

.

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

fliang: Describe MIDIs 127 iquantized pitches, isomorphic to musicXML

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

fliang: Define meter

Eck and Lapalme [32] 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 perodic measures of equal length [55]. Meter provides us with key information about musical structure which can be used to predict chord changes and repepetition boundaries [19].

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.

2.1 A primer on Western music theory

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.

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

Fig. 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 [104], 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.

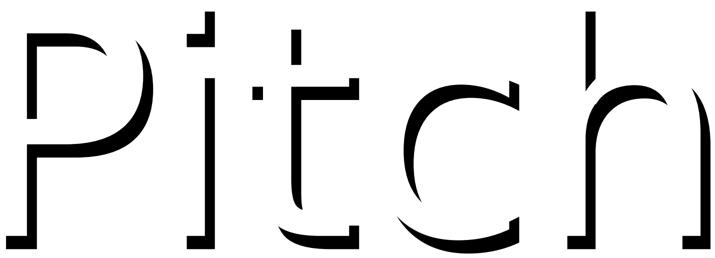


Fig. 2.2 Terhardt's visual analogy for pitch. Similar to how the viewer of this figure may percieve contours not present, pitch describes subjective information received by the listener even when physical frequencies are absent.

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 percieved on a logarithmic scale.

Most Western music is based on the *twelve-note chromatic scale*, which divides an *octave* into twelve distinct frequencies. The *tuning system* employed dictates the precise intervals between subdivisions, with *equal temperament tuning* (all subdivisions are equally spaced on a logarithmic scale) presently the most widely used method [28].

fliang: Talk about well-tempered tuning and bach

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.

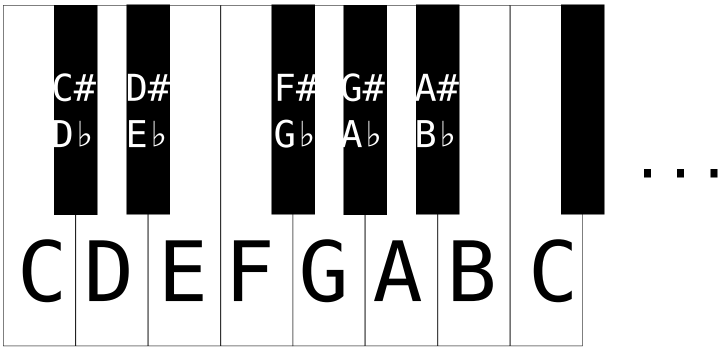


Fig. 2.3 Illustration of an octave in the 12-note chromatic scale on a piano keyboard.

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

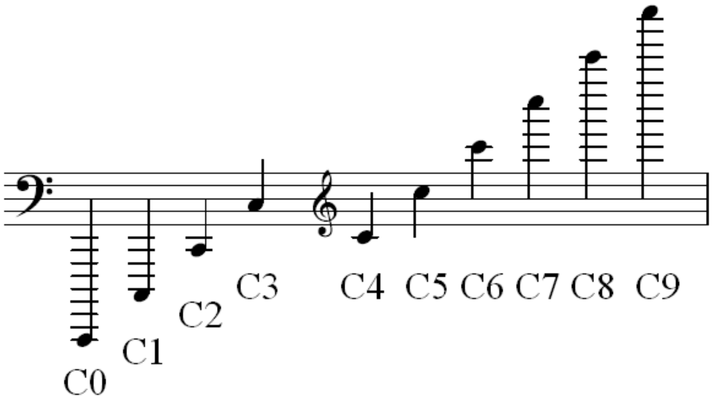


Fig. 2.4 Scientific pitch notation and sheet music notation of *C* notes at ten different octaves.

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.

2.1.2 Key signature

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 is built.

A basic concept within tonal music is *mode*, which defines a subset of pitch classes that are "in key" with respect to the tonic. **??** illustrates the pitch intervals between adjacent pitch classes within two important modes: the *major* (Ionian, I) and *minor* (Aeolian, VI) modes. 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[42]. The pitches in a scale can be found by starting at the tonic and successively offsetting by the given pitch intervals.

2.1.3 Polyphony, chords, and chord progressions

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

Chords: basic units for representing simultaneously sounding notes

A *chord* is a collection of three or more pitches all sounding simultaneously[90]. In Western classical music, they typically consist of a *root note* whose pitch class forms a base from which successive notes are built upon. The intervals between the pitch classes in a chord are commonly labeled using *qualities*, which are invariant across octaves. Different realizations of the same chord (e.g. octave choices for each pitch class) are called *voicings*.

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

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

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[42]

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[90]. 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 progresssions 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 $I - ii - IImaj7 - V$.

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.

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

2.2 Neural sequence probability modeling

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

2.2.1 Neurons: the basic computation unit

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

$$f(\boldsymbol{x}) \coloneqq \sigma(\langle \boldsymbol{w}, \boldsymbol{x} \rangle) \tag{2.1}$$

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

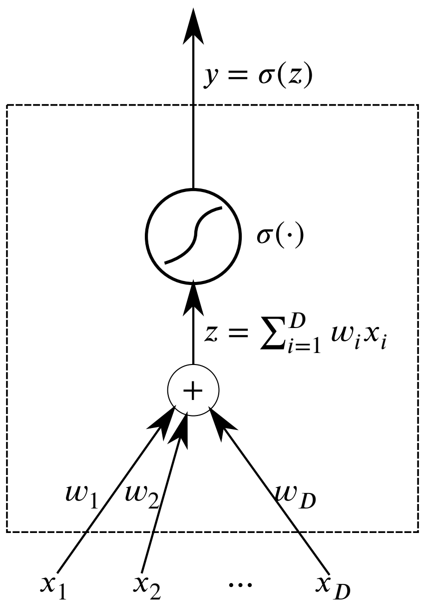


Fig. 2.6 A single neuron first computes an activation z and then passes it through an activation function $\sigma(\cdot)$

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*.

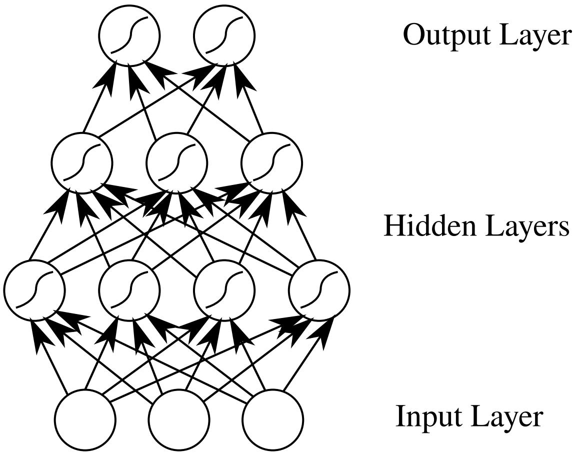


Fig. 2.7 Graph depiction of a feedforward neural network with 2 hidden layers

Although a single hidden layer is theoretically sufficient for a universal function approximator[25], 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[11].

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

$$\sigma\left(\boldsymbol{W}_{L,L-1}\sigma\left(\boldsymbol{W}_{L-1,L-2}\cdots\sigma\left(\boldsymbol{W}_{2,1}\boldsymbol{x}\right)\cdots\right)\right) \tag{2.2}$$

After removing the activation functions σ, we are left with

$$\boldsymbol{W}_{L,L-1}\boldsymbol{W}_{L-1,L-2}\cdots\boldsymbol{W}_{2,1}\boldsymbol{x} = \boldsymbol{x} = \tilde{\boldsymbol{W}}\boldsymbol{x} \tag{2.3}$$

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

2.2.3 Recurrent neural networks

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

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

This shortcoming motivates *recurrent neural networks* (RNNs), which generalize feedforward networks by introducing time-delayed recurrent connections between hidden layers (Elman networks [35]) or from the output layers to the hidden layers (Jordan networks [65]). Mathematically, an (Elman-type) RNN is a discrete time dynamical system commonly parameterized as:

$$\boldsymbol{h}_t = \boldsymbol{W}_{xh} \sigma_{xh} \left(\boldsymbol{x}_t \right) + \boldsymbol{W}_{hh} \sigma_{hh} \left(\boldsymbol{h}_{t-1} \right) \tag{2.4}$$

$$\boldsymbol{y}_t = \boldsymbol{W}_{hy} \sigma_{hy} \left(\boldsymbol{h}_t \right) \tag{2.5}$$

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

fliang: Why do we use Elman

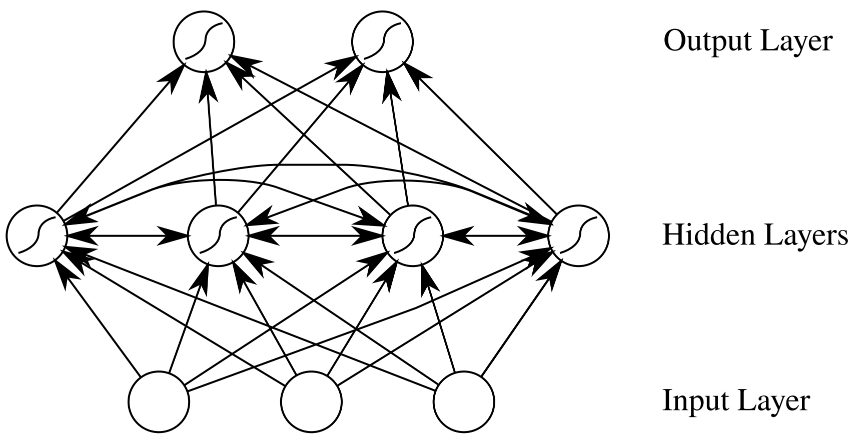


Fig. 2.8 Graph representation of an Elman-type RNN.

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

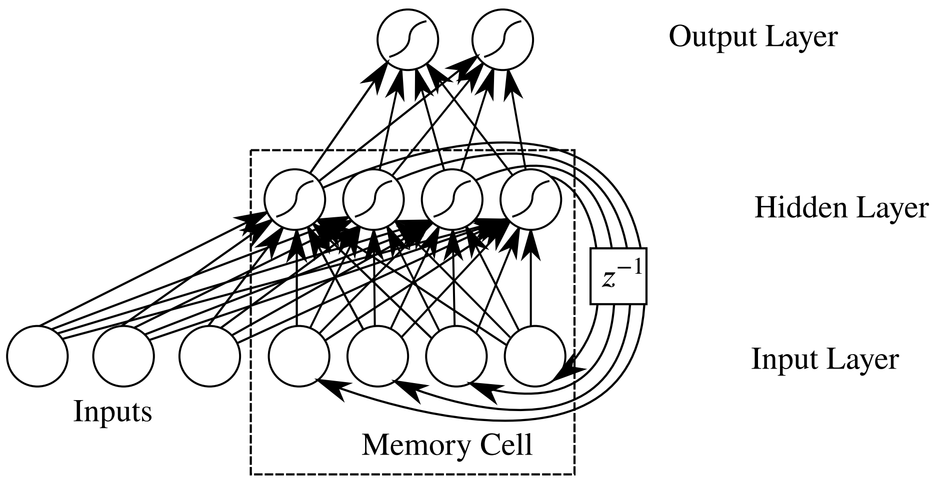


Fig. 2.9 Equivalent formulation of an Elman-type RNN treating the time-delayed hidden state $\boldsymbol{h}_{t-1}$ as additional inputs to a feedforward network

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 $\boldsymbol{y}_t$ and $\boldsymbol{h}_t$ are computed from $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$, 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 $\boldsymbol{x}_t$
- The previous hidden state $\boldsymbol{h}_{t-1}$

Outputs

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

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

Unrolling into a DAG

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

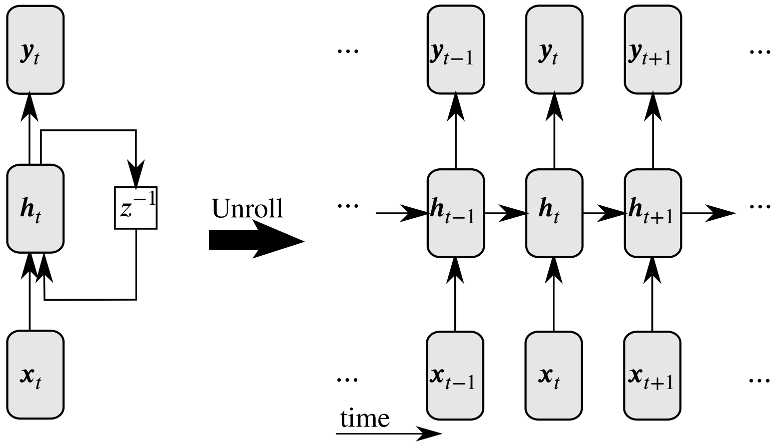


Fig. 2.10 Signal flow diagram representation of a single-layer RNN and its corresponding directed acyclic graph after unrolling

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

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

Stacking memory cells to form multi-layer RNNs

Since the RNN outputs $\boldsymbol{y}$ also form a sequence with the same length as the inputs $\boldsymbol{x}$, 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 **??** .

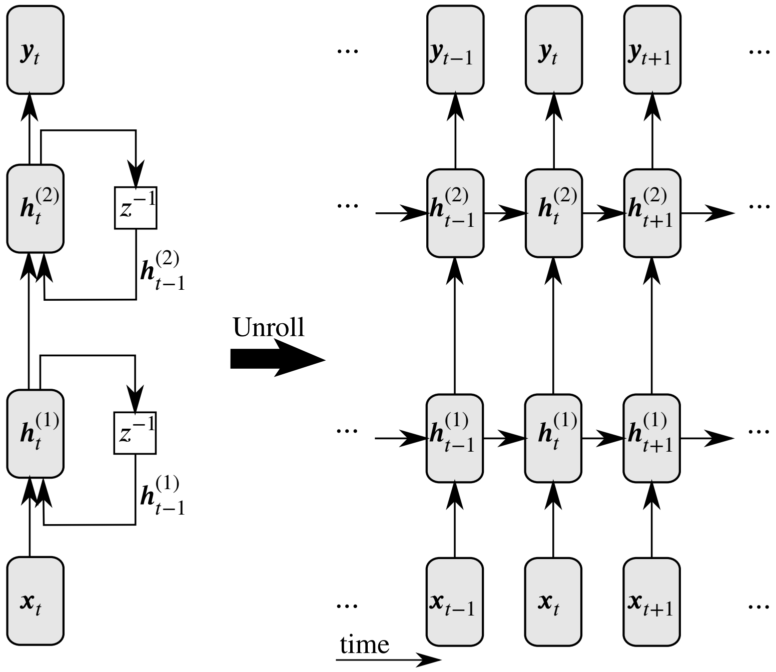


Fig. 2.11 Template and unrolling of a stacked 2-layer RNN

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 $\boldsymbol{x}_{t-1}$ to $\boldsymbol{y}_t$ hence the conditional independence $\boldsymbol{y}_t \perp\!\!\!\perp \boldsymbol{x}_{t-1} | \boldsymbol{h}_t^{(1)}$ is satisfied. However, in **??** there are multiple paths from $\boldsymbol{x}_{t-1}$ to $\boldsymbol{y}_t$ (e.g. passing through either $\boldsymbol{h}_{t-1}^{(2)} \to \boldsymbol{h}_t^{(2)}$ or $\boldsymbol{h}_{t-1}^{(1)} \to \boldsymbol{h}_t^{(1)}$) through which information may flow. Additionaly, the hidden state transitions occur on two seperate 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 $\boldsymbol{x}_{1:T}$, $\boldsymbol{y}_{1:T}$, and $\boldsymbol{h}_{0:T}$:

1. Modified Markov assumption:

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

2. Hidden State Stationarity:

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

3. Output Stationarity:

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

4. Output independence:

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

These assumptions imply the sequential factorization:

$$\begin{aligned}
& P(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{h}_0, \boldsymbol{x}_{1:T}) && (2.10)\\
&= P(\boldsymbol{y}_{1:T}|\boldsymbol{h}_{0:T}, \boldsymbol{x}_{1:T}) P(\boldsymbol{h}_{1:T}|\boldsymbol{h}_0, \boldsymbol{x}_{1:T}) && (2.11)\\
&= \left(\prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t)\right) P(\boldsymbol{h}_{1:T}|\boldsymbol{h}_0, \boldsymbol{x}_{1:T}) \qquad \textbf{??} && (2.12)\\
&= \left(\prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t)\right)\left(\prod_{t=1}^{T} P(\boldsymbol{h}_t|\boldsymbol{h}_{0:t-1}, \boldsymbol{x}_{1:t})\right) && (2.13)\\
&= \left(\prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t)\right)\left(\prod_{t=1}^{T} P(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)\right) \qquad \textbf{??} && (2.14)\\
&= \prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t, \boldsymbol{x}_t) P(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) && (2.15)\\
& && (2.16)
\end{aligned}$$

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

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

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

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

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

fliang: Draw PGM

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

fliang: ref

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

fliang: Define V to be the vocabulary

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

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

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

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 $P(\boldsymbol{y}_t|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$ to mean the Boltzmann distribution parameterized by the scores $f_y(f_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}))$.

Note the similarity between ?? –?? and the assumptions for Hidden Markov models [89]. Discrepancies are due to the presence of an input sequence $\boldsymbol{x}_{1:T}$ in our sequence-to-sequence model.

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

2.2.5 Training RNNs: backpropogation through time

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

One approach for computing the necessary gradients is *backpropogation through time* (BPTT)[48], an adaptation of the backpropogation algorithm[70, 93] to the unrolled RNN computation graph. We can apply the chain rule to the unrolled RNN's computation graph in **??** to obtain

$$\frac{\partial \mathscr{E}}{\partial \boldsymbol{\theta}} = \sum_{1 \leq t \leq T} \frac{\partial \mathscr{E}_t}{\partial \boldsymbol{\theta}} \tag{2.20}$$

$$\frac{\partial \mathscr{E}_t}{\partial \boldsymbol{\theta}} = \sum_{1 \leq k \leq t} \left(\frac{\partial \mathscr{E}_t}{\partial \boldsymbol{y}_t} \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{\theta}} \right) \tag{2.21}$$

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} = \prod_{t \geq i > k} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} = \prod_{t \geq i > k} \boldsymbol{W}_{hh}^\intercal \operatorname{diag}\left(\sigma'_{hh}(\boldsymbol{h}_{i-1})\right) \tag{2.22}$$

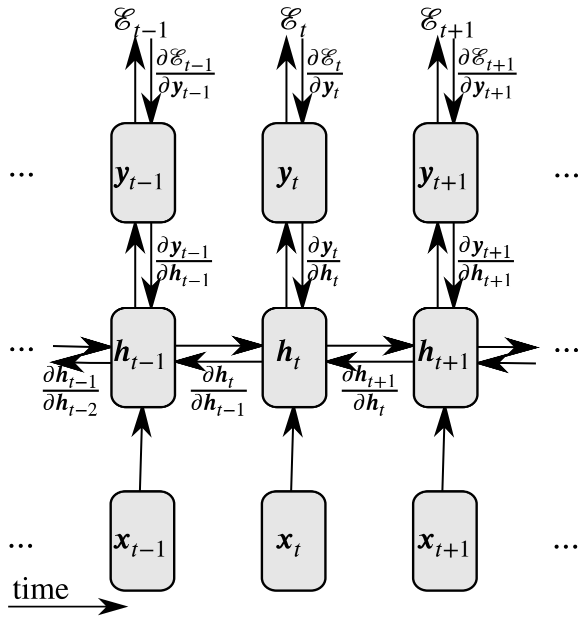


Fig. 2.12 The gradients passed along network edges during BPTT.

?? expresses how the error $\mathscr{E}_t$ at time t is a sum of *temporal contributions* $\frac{\partial \mathscr{E}_t}{\partial \boldsymbol{y}_t} \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{\theta}}$ measuring how θ's impact on $\boldsymbol{h}_k$ affects the cost at some future time $t > k$. The factors in **??** measures the affect of the hidden state $\boldsymbol{h}_k$ on some future state $\boldsymbol{h}_t$ where $t > k$ and can be interpreted as transferring the error "in time" from step t back to step k [85].

Vanishing/exploding gradients

Unfortunately, naive implementations of **??** and **??** oftentimes suffers from two well known problems: the *vanishing gradient* and *exploding gradient*[12]. 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. [85], let $\|\cdot\|$ be any submultiplicative matrix norm (e.g. Frobenius, spectral, nuclear, Shatten p-norms). Without loss of generality, we will use the *operator norm* defined as

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

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

From submultiplicativity, we have that for any k

$$\left\| \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{h}_{k-1}} \right\| \leq \|\boldsymbol{W}_{hh}^\intercal\| \| \operatorname{diag}\left(\sigma'_{hh}(\boldsymbol{h}_{k-1})\right) \| \leq \gamma_{\boldsymbol{W}} \gamma_\sigma \tag{2.24}$$

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

$$\begin{aligned}
\gamma_\sigma &\coloneqq \sup_{h \in \mathbb{R}^n} \| \operatorname{diag}\left(\sigma'_{hh}(\boldsymbol{h})\right) \| && && (2.25) \\
&= \sup_{h \in \mathbb{R}^n} \max_i \sigma'_{hh}(\boldsymbol{h})_i && \text{Operator norm of diag} && (2.26) \\
&= \sup_{x \in \mathbb{R}} \sigma'_{hh}(x) && \sigma_{hh} \text{ acts elementwise} && (2.27)
\end{aligned}$$

Substituting back into **??** , we find that

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

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

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

$$\gamma_{\boldsymbol{W}} < \frac{1}{\gamma_\sigma} \tag{2.29}$$

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

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

2.2.6 Long short term memory: solving the vanishing gradient

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

The *long short term memory* (LSTM) memory cell was proposed by Hochreiter and Schmidhuber [60] as a method for dealing with the vanishing gradient problem. It does so by enforcing *constant error flow* on **??** , that is

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

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

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

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

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

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

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

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

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

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

$$\boldsymbol{i}_t = \text{sigmoid}(\boldsymbol{W}_{xi}\boldsymbol{x}_t + \boldsymbol{W}_{yi}\boldsymbol{y}_{t-1} + \boldsymbol{b}_i) \tag{2.31}$$

$$\boldsymbol{o}_t = \text{sigmoid}(\boldsymbol{W}_{xo}\boldsymbol{x}_t + \boldsymbol{W}_{yo}\boldsymbol{y}_{t-1} + \boldsymbol{b}_o) \tag{2.32}$$

$$\boldsymbol{f}_t = \text{sigmoid}(\boldsymbol{W}_{xf}\boldsymbol{x}_t + \boldsymbol{W}_{yf}\boldsymbol{y}_{t-1} + \boldsymbol{b}_f) \tag{2.33}$$

$$\boldsymbol{h}_t = \boldsymbol{f}_t \odot \boldsymbol{h}_{t-1} + \boldsymbol{i}_t \odot \tanh(\boldsymbol{W}_{xh}\boldsymbol{x}_t + y_{t-1}\boldsymbol{W}_{yh} + \boldsymbol{b}_h) \tag{2.34}$$

$$\boldsymbol{y}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{h}_t) \tag{2.35}$$

where $\odot$ denotes elementwise multiplication of vectors.

Notice that the gates ($\boldsymbol{i}_t$, $\boldsymbol{o}_t$, and $\boldsymbol{f}_t$) controlling flow in and out of the CEC are all time varying. This can be interpreted as a mechanism enabling LSTM to explicitly learn which error signals to trap in the CEC and when to release them [60], enabling error signals to potentially be transported across long time lags.

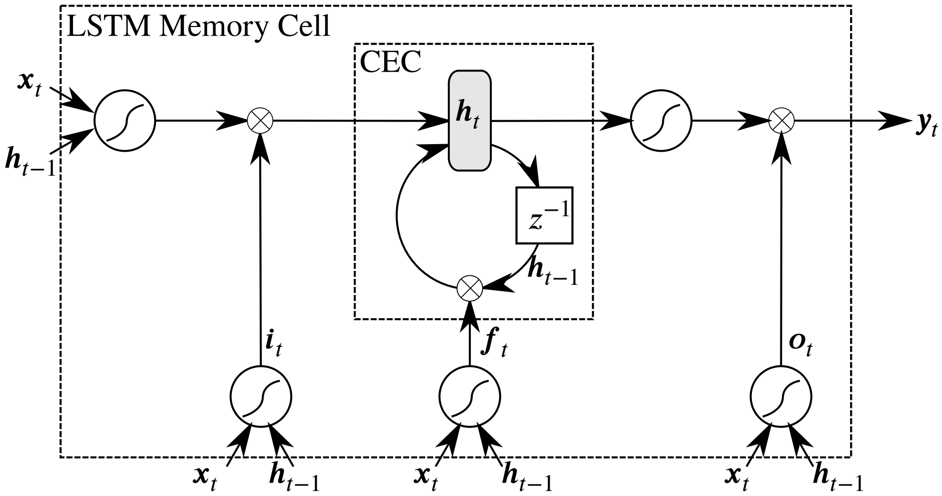


Fig. 2.13 Schematic for a single LSTM memory cell. Notice how the gates $\boldsymbol{i}_t$, $\boldsymbol{o}_t$, and $\boldsymbol{f}_t$ control access to the constant error carousel (CEC).

Some authors define LSTMs such that $\boldsymbol{h}_t$ is not used to compute gate activations, referring to the case where $\boldsymbol{h}_t$ is connected as "peephole connections"[45]. 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* [75, 85] 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* [96] can be applied to the connections between memory cells in a stacked RNN to regularize the learned features to be more robust to noise [118]. Additionally, *batch normalization*[64] can also be applied to to the memory cell hidden states to reduce co-variate shifts, accelerate training, and improve generalization.

Finally, applications of RNNs to long sequences can incur a prohibitively high cost for a single parameter update[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*[113]. 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 Coutinho et al. [21], which termed the phrase "computational musicology." Since then, development modern tools have greatly aided research efforts. `music21` [23] is a Python programming environment for performing computations over musical composition data which has been utilized for a variety of computational musicology tasks ranging from hierarchical modelling of metrical structure [6], feature generation for downstream machine learning[24], and style classification [57].

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

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

There is a vast body of research dealing with music classification tasks, including: style classification [57, 26], automated harmonic analysis [81], information retrieval [74], and performer identification [97]. However, it is not straightforward to utilize work in this area to solve our research goals of music synthesis.

fliang: of what?

.

3.2 Models for automatic composition

Unlike classification, the other tasks (harmonization, completion, automatic composition) require synthesis of novel music. In a review by Toiviainen [107], automatic composition methods are broady classified as either symbolic (i.e. rule based) or connectionist (i.e. neural networks).

3.2.1 Symbolic rule-based methods

Symbolic methods are popular due to their high degree of interpretability. As described by Todd [106], symbolic methods enable composers to write down the compositional rules employed in their own creative process and then use a computer to execute these instructions, enabling assessment of whether the results of the rules held artistic merit. This approach has been prevalent in automatic composition since the 1960s [106].

CHORAL [31] is one of the first rule based expert system for harmonising Bach chorales. It uses 350 manually defined rules as well as hand-tuned search heuristics. A later system called *Experiments in Music Intelligence* (EMI) [20] automatically extracted rules to build an augmented transition network[111]. In Cruz-Alcázar and Vidal-Ruiz [22], grammatical inference is used to learn regular grammars over chord progressions for modelling musical style. Tsang and Aitken [108] applies constraint logic programming for generating harmonizations which satisfy certain harmonic constraints.

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

3.2.2 Connectionist methods

Connectionism, also known as parallel distributed processing, is performed by a collection of several simple processing units connected in a network and acting in cooperation [54]. This shift in paradigm replaces strict rule-following behavior with regularity-learning and generalization [29].

Neural networks have been previously applied to music with varying degrees of success[53]. The earliest connectionist music models utilized note-level Jordan RNNs on melody generation and harmonization tasks [105, 106, 13].

fliang: say more here

A landmark connectionist system is Mozer's CONCERT [78], 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 [94] and chords use a distributed embedding originally trained for style classification [68]. The model passes objective evaluations by faithfully reproducing scales but "while the local contours made sense, the pieces were not musically coherent, lacking thematic structure and having minimal phrase structure and rhythmic organisation" (Mozer [78]).

Boulanger-Lewandowski et al. [14] proposed the RNN-RBM. a time-varying RBM with hidden units evolving over time according to a RNN, to model polyphonic music on a piano roll representation. However, training the RNN-RBM requires an expensive contrastive divergence sampling step at each timestep and a nontrivial Hessian-free optimisation routine. Furthermore, the authors quantized music to eighth-notes. In contrast, our work uses the well-understood truncated BPTT algorithm for training and quantizes to sixteenth-notes to achieve two-times higher time resolution.

Lyu [73] extended the RNN-RBM[14] 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 [58], which combines connectionist neural networks with formal rules to specifically harmonize Bach chorales. It implements a domain-specific processing pipeline consisting of:

1. Harmonic modelling: Predict harmonic skeleton (i.e. Roman numerals quantized to quarter-notes) using a neural network

2. Expand each Roman numeral to chords using formal rules

3. Ornamentation: add eighth-notes using formal rules

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

fliang: Define this task

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

MELONET [36] builds on top of HARMONET's harmonic modelling to construct chorale partitas (i.e. variations where one of the parts is varied in a harmonically believable way). It first introduced multiple ideas which have been rediscovered in recent years, including:

1. Delayed update units to model multiple timescales (described again in Clockwork RNNs [66])

2. Use of Resilient Propagation (RProp) [91] for training (described again in Liu and Ramakrishnan [71])

Additionally, MELONET utilizes a motif classification neural network to explicitly force motifs to appear multiple times within a partita. Follow up work by Hornel and Ragg [63] extends MELONET to use a distributed representation for motifs and a genetic algorithm for training. While MELONET introduces many novel ideas, its limited training set size (16 Pachelbel chorales [62]) and domain-specific architecture limit the generalizability of results.

fliang: Better analyze this

CHIME [37] adopted the Jordan RNN from Todd [106] to add a second training phase using actor-critic reinforcement learning [103]. The critic is constructed using a collection of "music rules," enabling incorporation of prior knowledge.

3.2.4 LSTM music synthesis models

Prior work has demonstrated LSTM possesses many properties desirable for music applications. Their superiority over traditional RNNs has been well documented[44]. They can learn to count and measure time intervals between events spaced arbitrarily far apart in time [45], a property N-gram language models do not possess. Gers et al. [47] demonstrated LSTM learning to produce self-sustaining oscillations at a regular frequency, suggesting that they are capable of discovering periodic structure. Franklin [41] evaluates various RNN architectures on variety of music tasks and concludes: "while we have found a task that challenges a single LSTM network, we do not believe that any other recurrent networks we have used would be able to learn these songs."'

One of the first applications of LSTMs to music was by Eck and Schmidhuber [33, 34]. Using a LSTM to model blues chord progressions and another to model melody lines given chords, the authors reported that LSTM can learn long term music structure such as repeated motifs can be learned without explicit modelling (e.g. MELONET). However, the music representation quantized to eighth notes, used considered pitch classes without accounting for octaves, and limited the model to 12 possible chords. Additionally, there was "no explicit way to determine when a note ends," prohibiting discrimination between four consecutive articulations of a note at the same pitch from a single note held for four timesteps. In contrast, our model accounts for the octaves in addition to pitch class, does not restrict the possible chords, operates at twice the time resolution, and also models when a note ends.

More recently, Sturm et al. [99, 100] 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)[18], which constrains the input and forget gates to sum to 1. Mikolov et al. [76] proposed the structurally constrained RNN (SCRN), a simple architecture achieving comparable performance to LSTMs. Of most relevance to music, Koutnik et al. [66] proposed the clockwork RNN for explicitly modelling phenomena occurring at multiple timescales by updating different blocks of the hidden state at different periods. Whether these differences matter is not definitive: Greff et al. [52] performed 5400 experiments on eight different architectures and found no significant difference in performance compared to the original LSTM architecture. Nayebi and Vitelli [80] 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 Tsaing's effective Boltzmann machine model [8]. Their model uses Boltzmann machines to enforce consistency within local contexts. As a result, their model is unable to capture long-range dependencies. Furthermore, they quantize to half-notes and only achieve 1/8 the time resolution of our model.

Allan and Williams [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.

Liu and Ramakrishnan [71] applied LSTMs to Bach chorales and reports significant gains using RProp instead of BPTT, a technique previously utilized by MELONET[36]. However, they erroneously use a mean squared error training criterion for a classification task, casting doubts on the validity of their experiments.

Brien and Roman [16] compared RNN models for Bach chorales and found clockwork RNNs to yield the lowest validation loss. However, their data format does not permit independent articulation of parts. More importantly, the performance margin between clockwork RNNs and LSTMs was very small (6.5 vs 6.75 cross-entropy loss) and their implementation resets the LSTM state when truncating gradients during BPTT, limiting the time-range of learned dynamics to be at most the sequence length.

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

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

Ada Lovelace [15]

4

Automatic composition with deep LSTM

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 [88] (i.e. no parallel fifths) and stylistic elements (i.e. voice leading) which can be use 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

chorale consists of a series of *phrases*: groupings of consecutive notes into a unit that has complete musical sense of its own[79]. It is well known

fliang: citep

that Bach denoted ends of phrases with fermatas

fliang: refer back to background

.

3. The Bach chorales have become a standardized corpus routinely studied by aspiring music theorists[112]

The Bach chorales, indexed by the Bach-Werke-Verzeichnis (BWV) numbering system[17], are conveniently provided by `music21`[23].

4.1.1 Preprocessing

Motivated by the transposition invariance of music and prior practice [78] [33] [39] [40], we first perform *key normalization*. The key signature of each score were firsanalyzed using the Krumhansl Schmuckler key-finding algorithm [67] and then transposed according to

fliang: Table XYZ

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

Next, we perform *time quantization* by aligning note start and end times to the nearest multiple of some minimum duration. Our model uses a minimum duration of one 1/16th note, exceeding the time resoltuions of [14] [33] by 2x, [58] by 4x, and [8] by 8x.

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

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

fliang: Is piano roll defined?

notation on 4.2 on page 55.

Corpus level analysis of preprocessing effects

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

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.

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

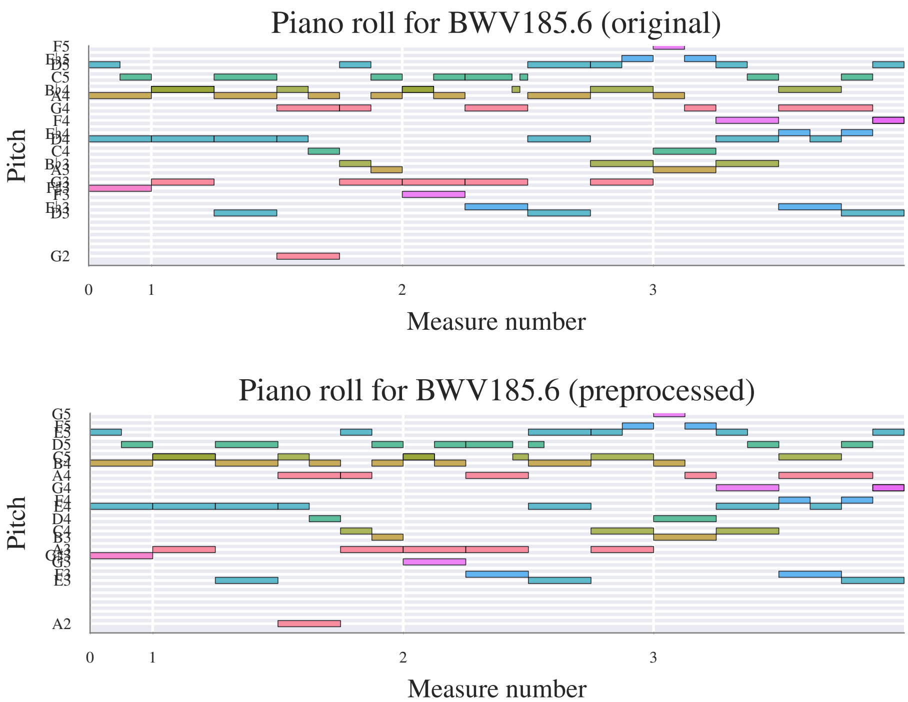


Fig. 4.2 Piano roll representation of the same 4 bars from **??** before and after preprocessing. Again, note the transposition to C-major and time-quantization occuring in the Soprano part.

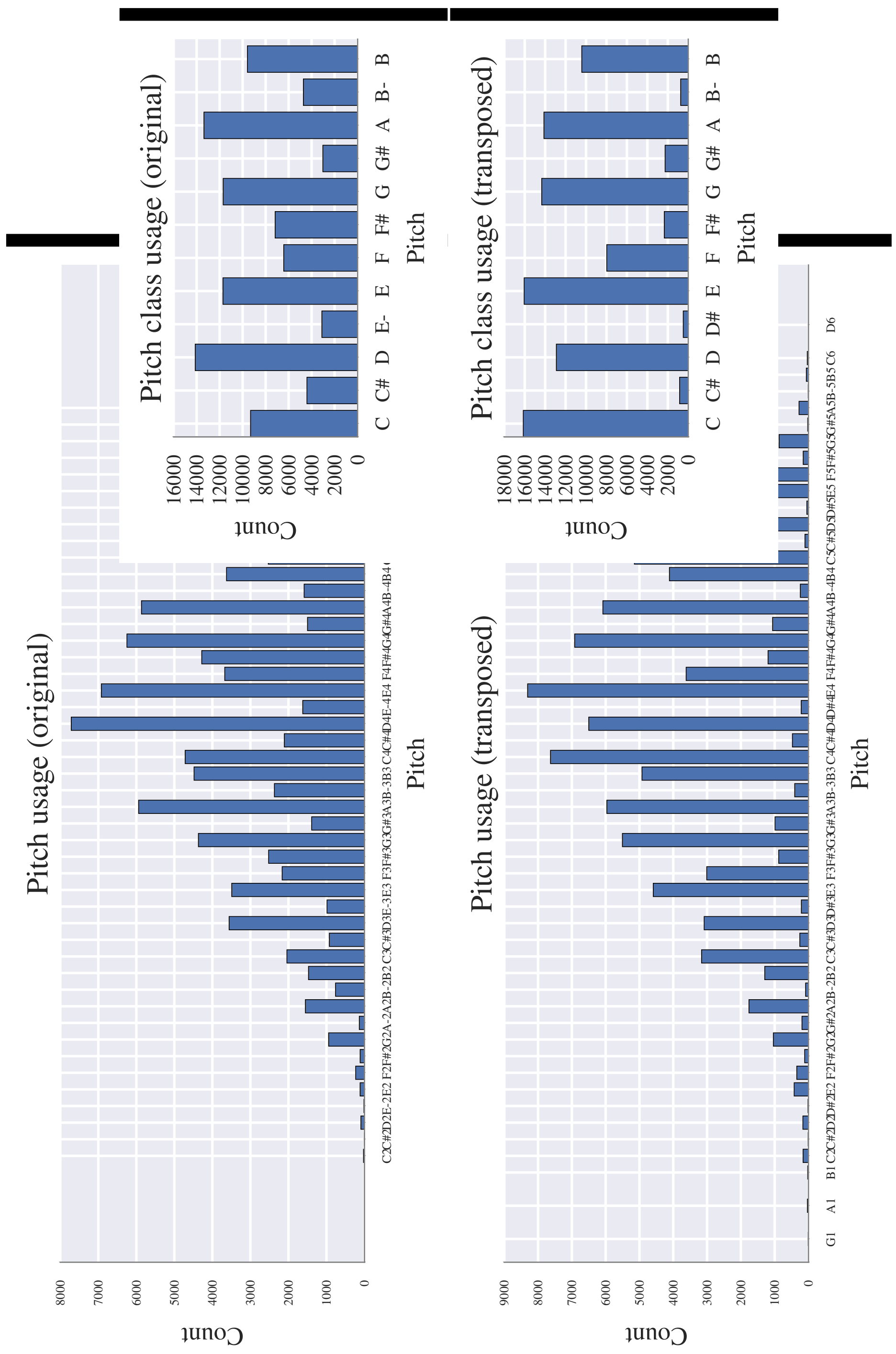


Fig. 4.3 Distribution of pitches used over Bach chorales corpus. Transposition has resulted in an overall broader range of pitches and increased the counts of pitches which are in key.

Fig. 4.4 Distribution of pitch classes over Bach chorales corpus. Transposition has increased the counts for pitch classes within the C-major / A-minor scales.

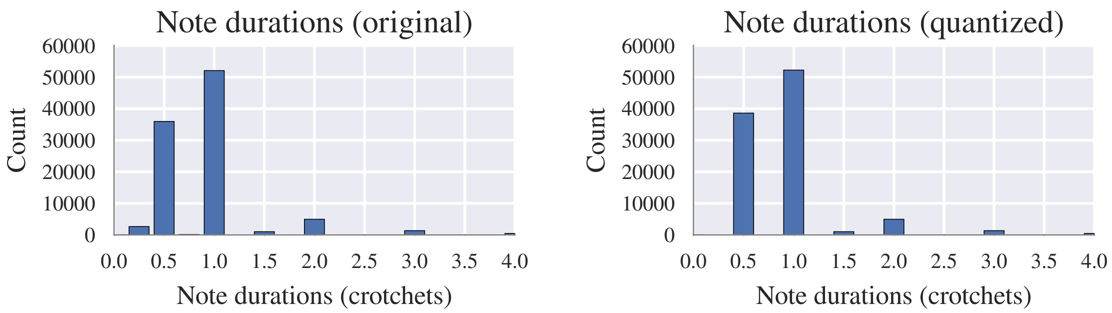


Fig. 4.5 Distribution of note durations over Bach chorales corpus. Quantization has minimal impact because of the high resolution (semiquavers) used.

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

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

fliang: Update plots... are they affected

.

4.1.2 Sequential encoding of musical data

After preprocessing of the scores, our next step is to encode music into a sequence of tokens amenable for processing by RNNs. One design decision is whether the tokens in the sequence are comprised of individual notes (as done in [78, 39**?**]) or larger harmonic units (e.g. chords [33, 14], "harmonic context" [3]). This tradeoff is similar to one faced in RNN language modelling where either individual characters or entire words can be used.

In contrast to most language models which operate at the word level, we choose to construct our models at the note level for several reasons. Firstly, the issue of multiple tokens in the sequence corresponding to the same instant of time in the represented music should not be problematic because LSTMs have been shown to be able to learn to implement precise timing and counting[47]. Additionally, the use of a note-level encoding partially mitigates the problem of out-of-vocabulary (OOV) tokens in two ways. Besides reducing the potential vocabulary size from $O(128^4)$ possible chords to $O(128)$ potential notes, the model is now able to capture harmonic relationships between notes within the LSTM model weights ($\boldsymbol{W}_{xx}$, $\boldsymbol{W}_{xh}$, $\boldsymbol{W}_{hh}$ in

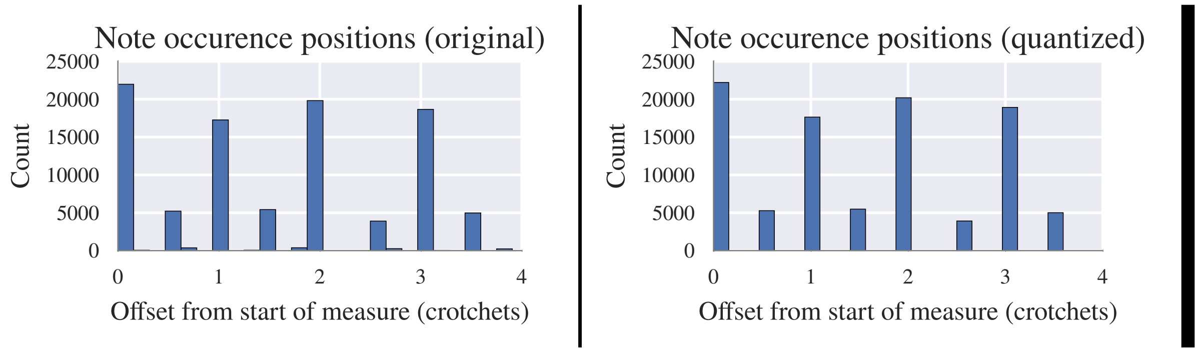


Fig. 4.6 Meter is minimally affected by quantization due to the high resolution used for time quantization.

fliang: reference

) and may generalize better to unseen chords. Furthermore, Graves [50] showed comparable performance between LSTM language models that operate on individual characters versus words (perplexities of 1.24 bits vs 1.23 bits per character respectively), suggesting that the choice is not too significant at least for English language modelling.

Similar to [106], we represent polyphonic scores using a localist frame-based representation where time is discretized into constant timestep *frames*. Frame based processing forces the network to learn the relative duration of notes, a counting and timing task which [47] demonstred LSTM is capable of. Consecutive frames are separated by a unique delimiter ("|||" in

fliang: Figure of score encoded in text

).

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

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

fliang: reference

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

Table 4.1 Statistics on the preprocessed datasets used throughout our study

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

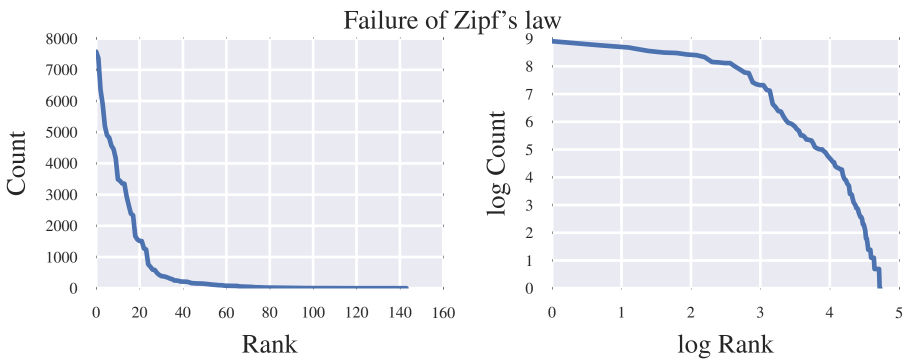


Fig. 4.7 Left: Token frequencies sorted by rank. Right: log-log plot where a power law distribution as predicted by Zipf's law would appear linear.

fliang: Figure of encoded score

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

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

fliang: Figure

) and end symbol ("END" in

fliang: Figure

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

Notice that our encoding is sparse: unarticulated notes are not encoded. It is also variable length as anywhere from zero to four (in the case of chorales, more for arbitrary polyphonic scores) notes. Finally, the explicit representation of tied notes vs articulated notes solves the problem plaguing [33][32] [71] [16] 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 [78] or harmonically-based [39] [68]. This is intentional and is motivated by numerous reports [10][11] suggesting that that a key ingredient in deep learning's success is its ability to learn good features from raw data.

4.2 Design and validation of a generative model for music

In this section, we describe the design and validation process leading to our generative model. Unlike many prior models for music data, we intentially avoid injection of domain-specific knowledge into our model architectures such as distinguishing between chords versus notes [58][78] [33] and explicitly modelling of meter [32] or motifs [36]. 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 [78], we will train the model to predict a distribution distribution over all possible tokens next $\boldsymbol{x}_{t+1}$ given the current token $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$. This is equivalent to setting the target sequence to be the input sequence delayed by one timestep: $\boldsymbol{y}_{1:T-1} = \boldsymbol{x}_{2:T}$ and $\boldsymbol{y}_T = \text{STOP}$.

fliang: Diagram for sequential prediction

.

fliang: Note similarity with language modeling

.

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

Note that our training criteria as written in

fliang: reference

uses the actual next token $\boldsymbol{x}_{t+1}$ as the recurrent input, even if the most likely prediction argmax $P(\boldsymbol{x}_{t+1} | \boldsymbol{h}_t, \boldsymbol{x}_t)$ differs. This is is referred to as *teacher forcing*[114] and is motivated by the observation that model predictions may not yet be correct during the early iterations of training. However, at inference the token generated from $P(\boldsymbol{x}_{t+1} | \boldsymbol{h}_t, \boldsymbol{x}_t)$ is reused as the previous input, creating a discrepancy between training and inference. Scheduled sampling [9] is a recently proposed alternative training method for resolving this discrepancy and may help the model better learn to predict using generated symbols rather than relying on ground truth to be always provided as input.

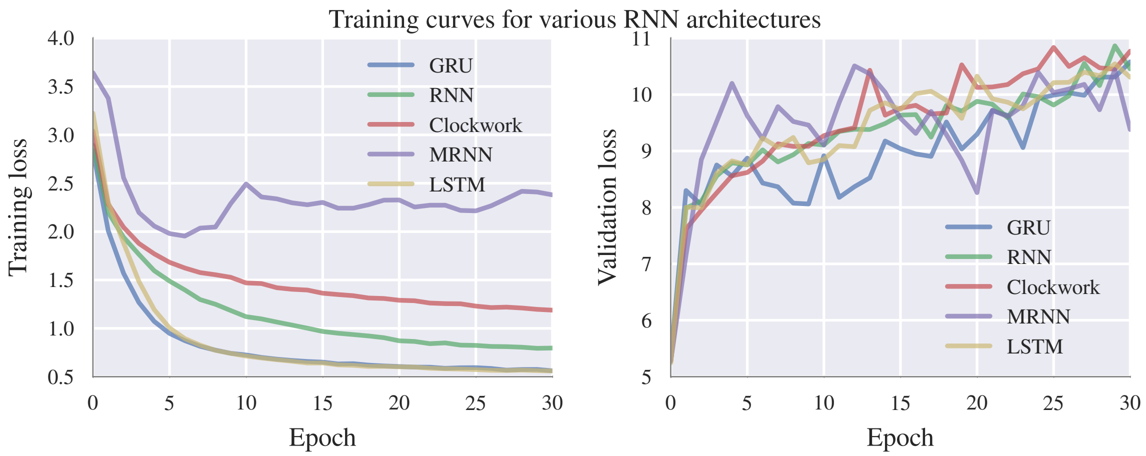


Fig. 4.8 LSTMs and GRUs yield the lowest training loss. Validation loss traces show all architectures exhibit signs of significant overfitting

4.2.2 Comparing memory cells for music data

Using theanets

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

fliang: Diagram of the model

In **??** , we compare various RNN architectures on our data. All models utilized a RNN with `num_layers=1`, `rnn_size=130`, `wordvec=64` and differed only in memory cell implementation. The clockwork RNN periods were set to $(1, 2, 4, 8, 16)$.

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

LSTMs and GRUs trained much faster and achieved lower training loss, suggesting higher capacity. [80] reports that LSTMs outperform GRUs in music applications, motivating our 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

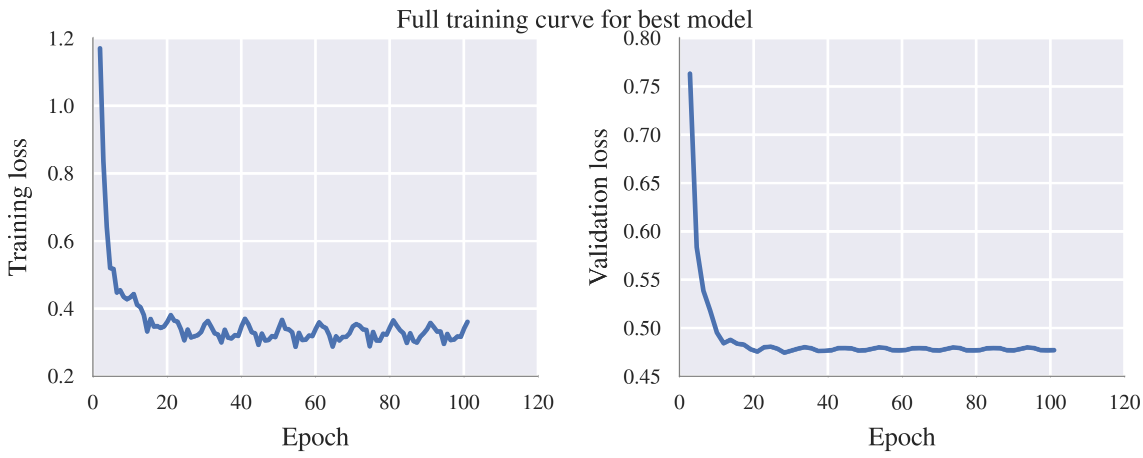


Fig. 4.9 Training curves for the overall best model. The periodic spikes correspond to resetting of the LSTM state at the end of a training epoch.

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 `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-propogation 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?

.

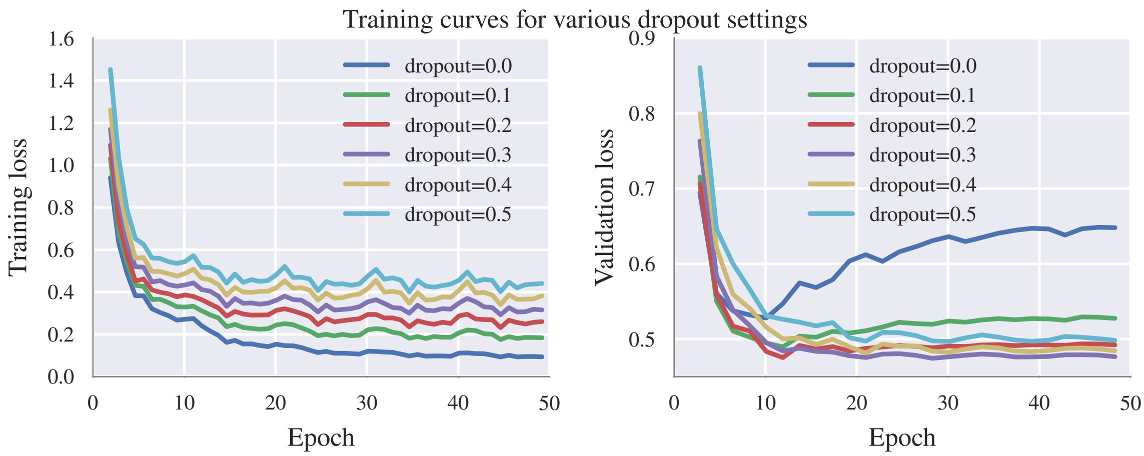


Fig. 4.10 Dropout acts as a regularizer, resulting in larger training loss but better generalization as evidenced by lower validation loss. A setting of `dropout=0.3` achieves best results for our model.

In the following sections, we investigate pertubations 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

Network capacity

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

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

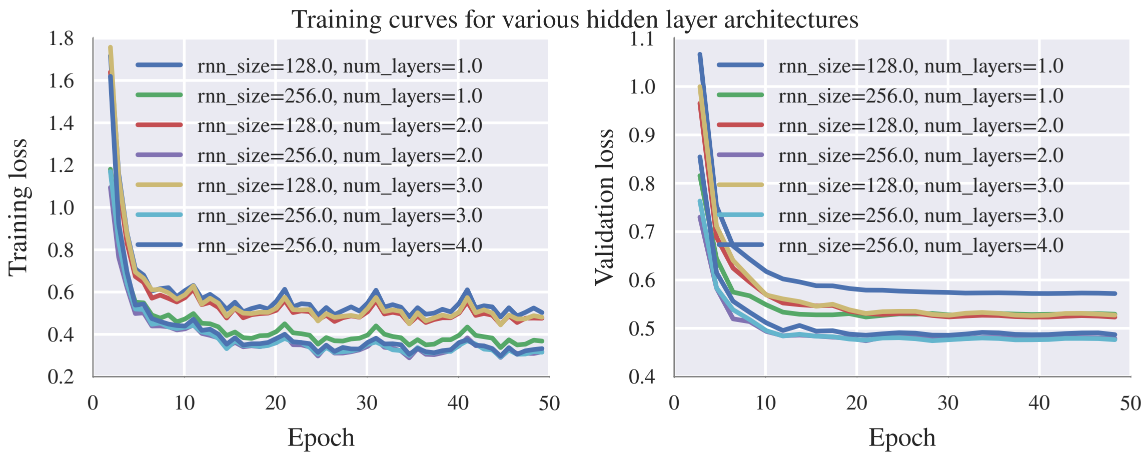


Fig. 4.11 `rnn_size=256` and `num_layers=3` yields lowest validation loss.

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

Network input parameters

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

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

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

  fliang: investigate further

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

4.3 Results

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

Fig. 4.12 Validation loss improves initially with increasing network depth but deteriorates after > 3 layers.

The baseline results obtained by N-grams are shown in 4.2 on the following page. We compare against two widely available language modelling software packages, KenLM [56] and SRILM [98]. KenLM implements an efficient moddified Kneser-Ney smoothing language model and while SRILM provides a variety of language models we choose choose to use the Good-Turing discounted language model for benchmarking against.

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 107), 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. [14] achieved -5.56 log likelihood and 33.12% accuracy on the symbolic prediction task. Note that the log likelihood is significantly lower than the -0.323 log likelihood implied by our model's cross-entropy loss. This is because the two results are not comparable as they utilize different input encodings with varying vocabulary sizes.

fliang: Compare [16]

.

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

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

fliang: Fill in CPU total when converged

Table 4.2 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

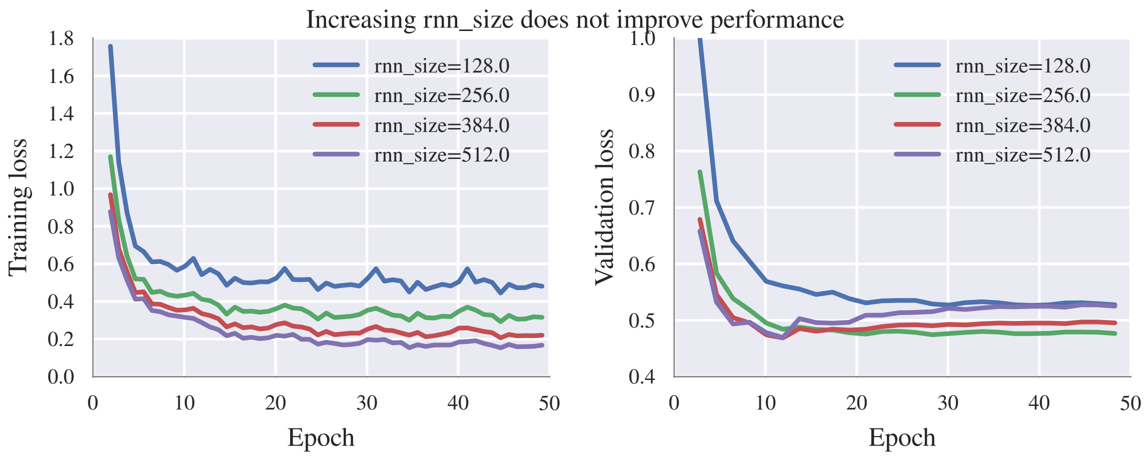


Fig. 4.13 Validation loss improves initially with higher-dimensional hidden states but deteriorates after > 256 dimensions.

Table 4.3 Timing results comparing training on CPUs vs GPUs

	Single Batch (seconds)		100 Epochs (seconds)
	mean	std	
CPU	4.287	0.311	??
GPU	0.513	0.001	5614

4.4 Other applications

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

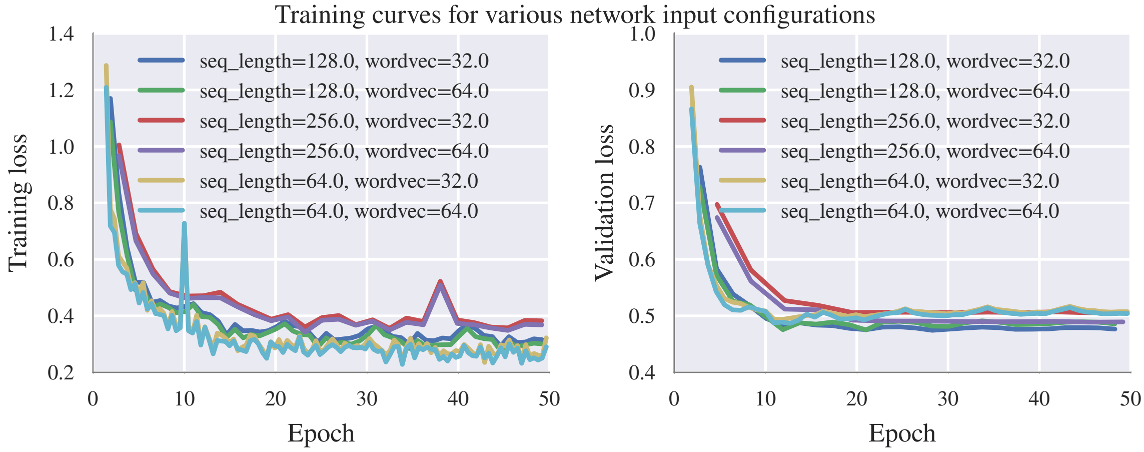


Fig. 4.14 `seq_length=128` and `wordvec=32` yields lowest validation loss.

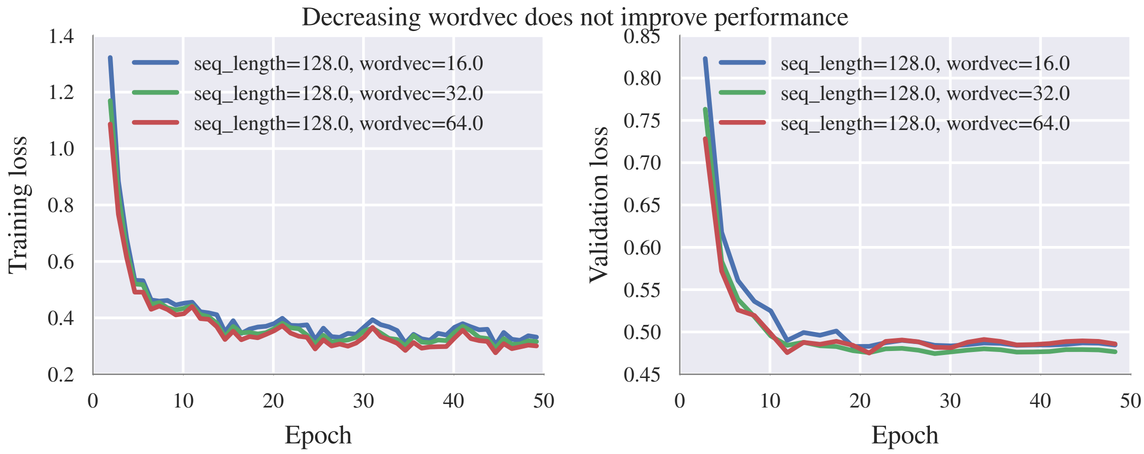


Fig. 4.15 Pertubations about `wordvec=32` do not yield significant improvements.

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

Hoffmann [61]

5 Chorale harmonization

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

Unlike automatic composition, in harmonization tasks we are given the entire sequence of notes for one or more parts. As one of the parts is now fixed, the model is no longer able to freely compose and harmonic deviations from the fixed parts will result in dissonances and conflicting expectations. This lack of accountability for future expectations is one of the failure modes of our models. One potential method for mitigating this is bidirectional LSTMs[51], 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 soprany melody. As described by Walter Piston [88]:

> 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 [88].

For a music student studying chorale harmonization, a common pedagogical exercise [27][88] 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. Attenpts to formalize the process using Shenkerian structural analysis [83] and symbolic methods such as generative grammars [69][116] 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, $\alpha^c \coloneqq [1, T] \setminus \alpha$, and w_α the tokens corresponding to the given parts. We are interested in finding

$$w^*_{1:L} = \operatorname*{argmax}_{w_{1:L}} P(w_{1:L} | w_\alpha) \tag{5.1}$$

“Clamp” the generative model and have it “fill-in” missing bits [59]. We can constrain the set of candidate sequences by first noting any solution $w\hat{_{1:L}}$ must satisfy $\hat{w}_\alpha = w_\alpha$. We can apply this constraint and greedily sample from our generative model to approximately solve the problem:

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

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

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

is true and justifies our model, the factorization

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

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 $\hat{w_{1:t-1}}$ 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

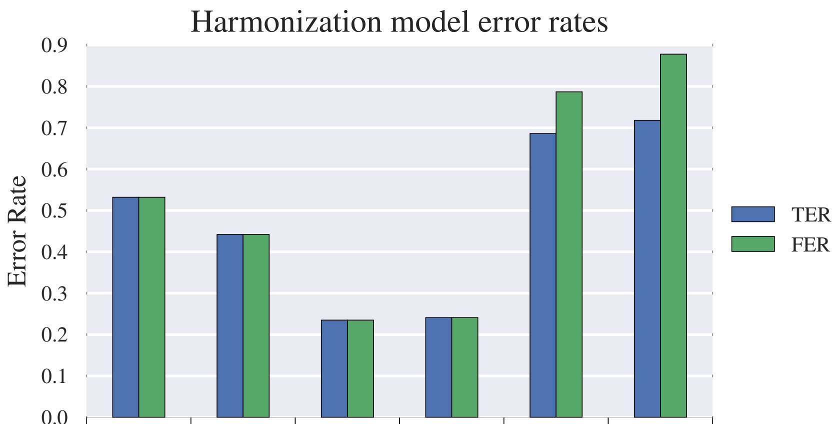
  fliang: mikolov

  that such an assumption can lead to very sensitive hidden state dynamics which are not robust to errors (i.e. when $\hat{w_{1:t-1}}$ 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. [72] would allow the model to partially recover from mistakes made during greedy action selection.

Furthermore, another potential issue with the proposed model is a discrepancy between the inputs provided during training (which are taken from actual data sequences regargless of model predictions) and sampling (where the inputs are generated by the model at the previous timestep). To resolve this discrepancy, Bengio et al. [9] proposed *scheduled sampling* as a curriculum learning strategy gradually transitioning sequence models to rely on their prior predictions as inputs and thereby learn state dynamics more resilient to erroneous inputs.

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



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

Fig. 5.1 Error rates for harmonization tasks

5.3 Datasets

We create datasets where one or more parts are masked:

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

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

5.4 Results

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

Fig. 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 acompanyments 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.

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

6

Large-scale subjective evaluation

Many prior studies [34, 3, 14, 73] evalute their success using either log likelihood or self-assessing some generated samples.

[87] addresses difficulty in quantitative evaluation, suggesting the use of a learned critic in a manner similar to GANs [49]. In a later report, [86] attribute difficulty in evaluation due to lack aim: algorithmic composition, design of compositional tools, and computational modelling of musical styles or music cognition all have different motivations and should thus be evaluated differently.

Following advice of [86], we identify our research motivation as building "computational models" for musical style and cognition. Specifically, our goal is to build a model which composes music in a style that is perceptually indistinguishable from Bach. To evaluate our results, we adapt Alan Turing's proposed "Imitation Game" [109] and carry out a large-scale musical Turing test.

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

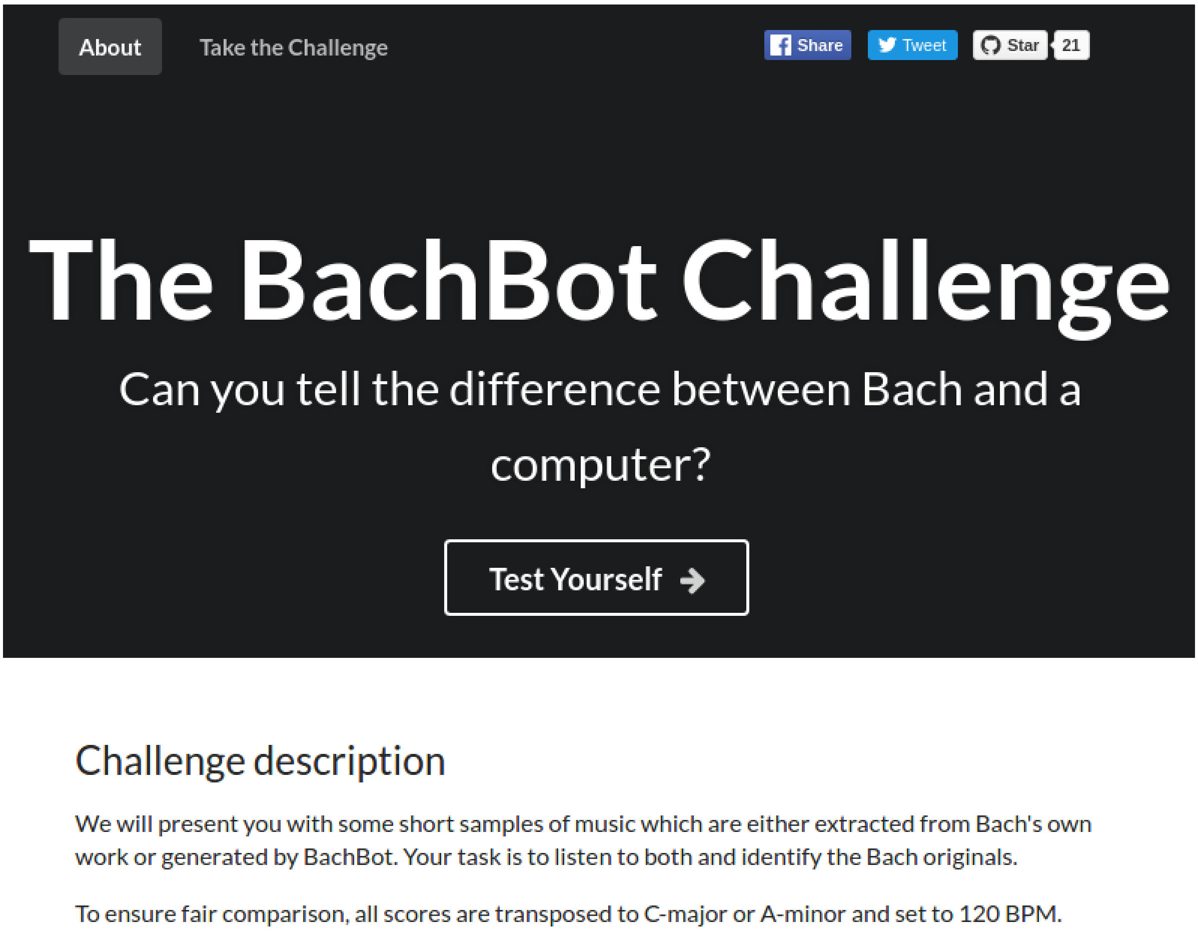


Fig. 6.1 The first page seen by a visitor of http://bachbot.com

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

6.1.2 User interface

The landing page for http://bachbot.com/ is shown in **??** .

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

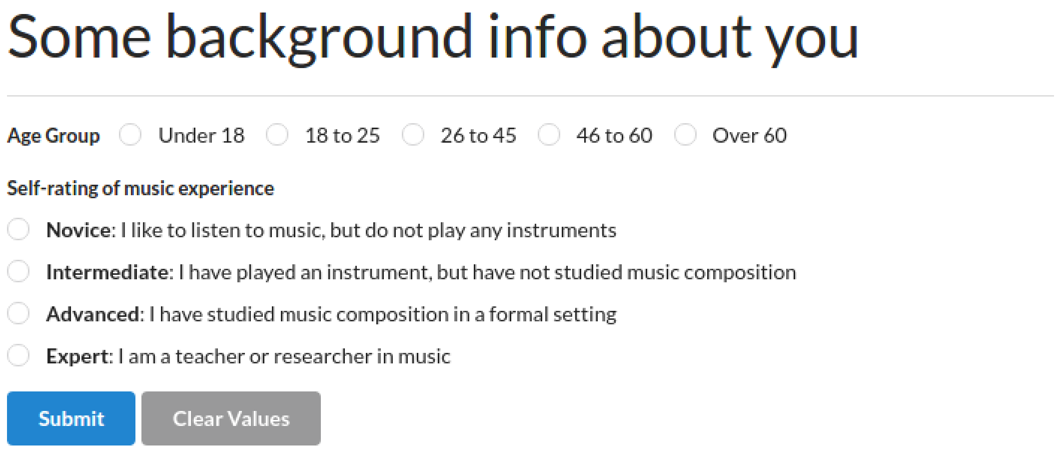


Fig. 6.2 User information form presented after clicking “Test Yourself”

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

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 sampls were paired with chorales randomly sampled from the corpus. The five question answered by any given participant were comprised of one S/A/T/B question chosen at random, one AT question, one ATB question, and two original compositions. See **??** for details.

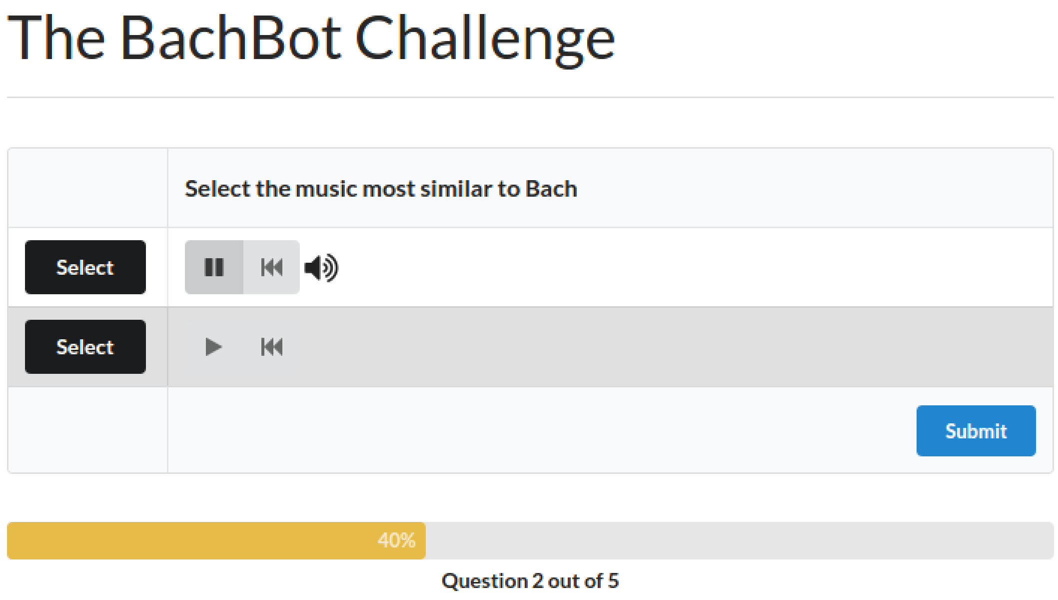


Fig. 6.3 Question response interface used for all questions

6.2 Results

6.2.1 Participant backgrounds and demographics

We recieved a total of

fliang: FILL THIS IN LAST

responses from

fliang: FILL THIS IN LAST

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

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

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

fliang: VERIFY LAST

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

fliang: VERIFY LAST

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

fliang: Collins only uses expert evaluators

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

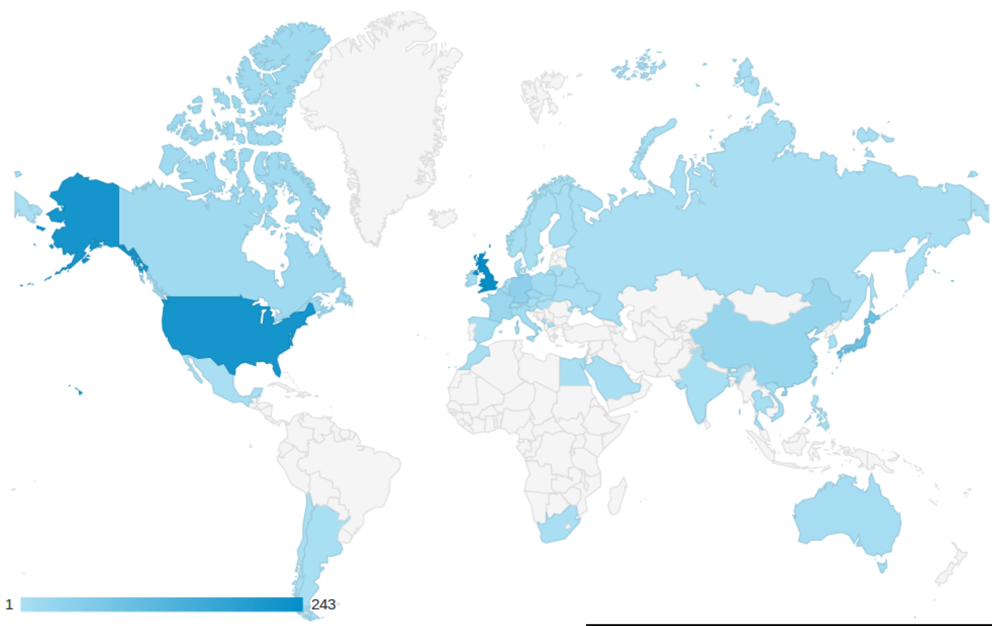
?? also shows that participants had more trouble discriminating entire compositions (SATB) than harmonizations (AT, ATB) where a subset of the parts have already been given. While this may seem counterintuitive, recall that the model in

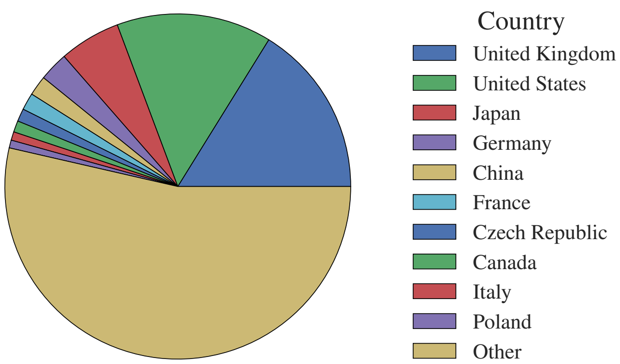
fliang: reference

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

fliang: cite

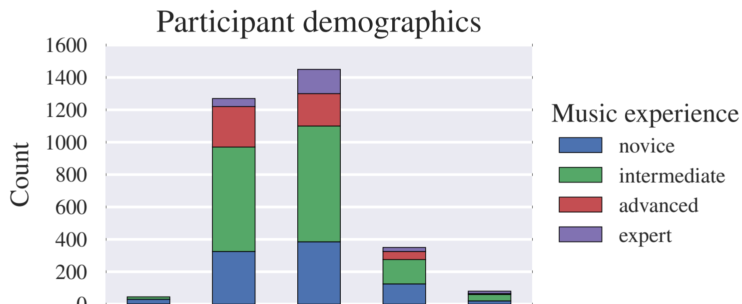
inputs) to mitigate this problem, which we leave for future work.





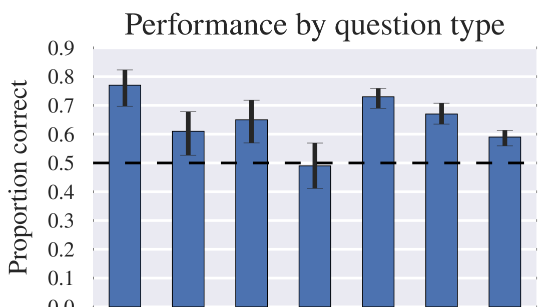
C[redacted]untry	Responses	Proportion
U[redacted]	243	16.0%
U[redacted]	218	15.0%
Ja[redacted]an	86	6.0%
Ge[redacted]rmany	41	3.0%
Ch[redacted]na	28	2.0%
Fr[redacted]nce	24	2.0%
Cz[redacted]echia	18	1.0%
Ca[redacted]nada	16	1.0%
Ita[redacted]y	12	1.0%
Po[redacted]and	11	1.0%
Ot[redacted]er	805	54.0%

Fig. 6.4 Geographic distribution of participants



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

Fig. 6.5 Demographics of participants



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

Fig. 6.6 responses-Mask

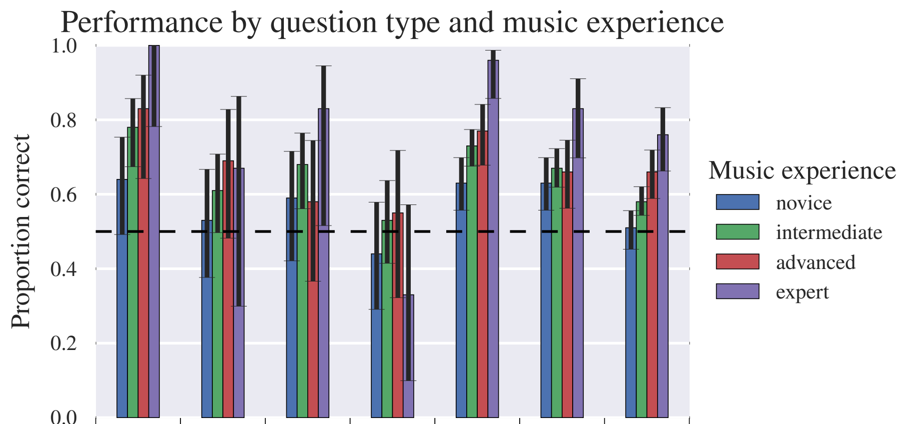
When only a only single part is composed by BachBot, we find the results vary significantly across different parts. Composing the soprano part proved to be the easiest to discriminate, an unsurprising result given that in chorale style music soprano parts are responsible for the melody

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%



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

Fig. 6.7 responses-mask-MusicExperience

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 **??**

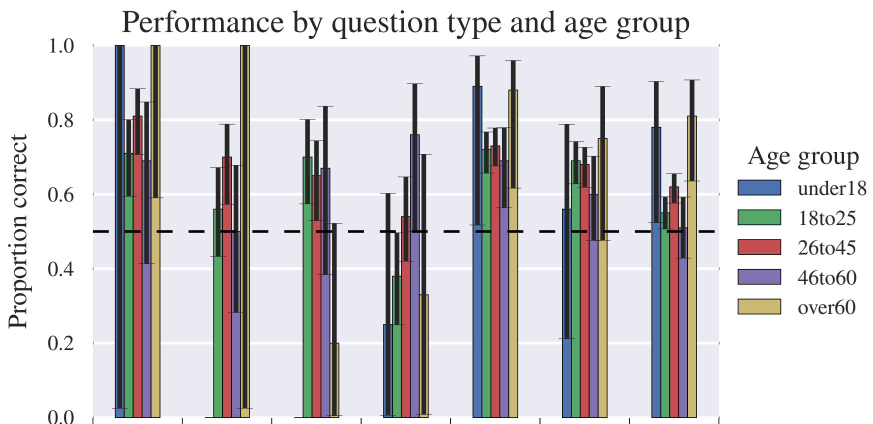
6.3 User feedback

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



	S	A	T	B	AT	ATB	SATB
under18	1.0	0.0	0.0	0.25	0.89	0.56	0.78
18to25	0.71	0.56	0.7	0.38	0.72	0.69	0.55
26to45	0.81	0.7	0.65	0.54	0.73	0.68	0.62
46to60	0.69	0.5	0.67	0.76	0.69	0.6	0.51
over60	1.0	1.0	0.2	0.33	0.88	0.75	0.81

Fig. 6.8 responses-mask-Agegroup

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. In contrast, 20%

fliang: UPDATE THIS NUMBER

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

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

[30]

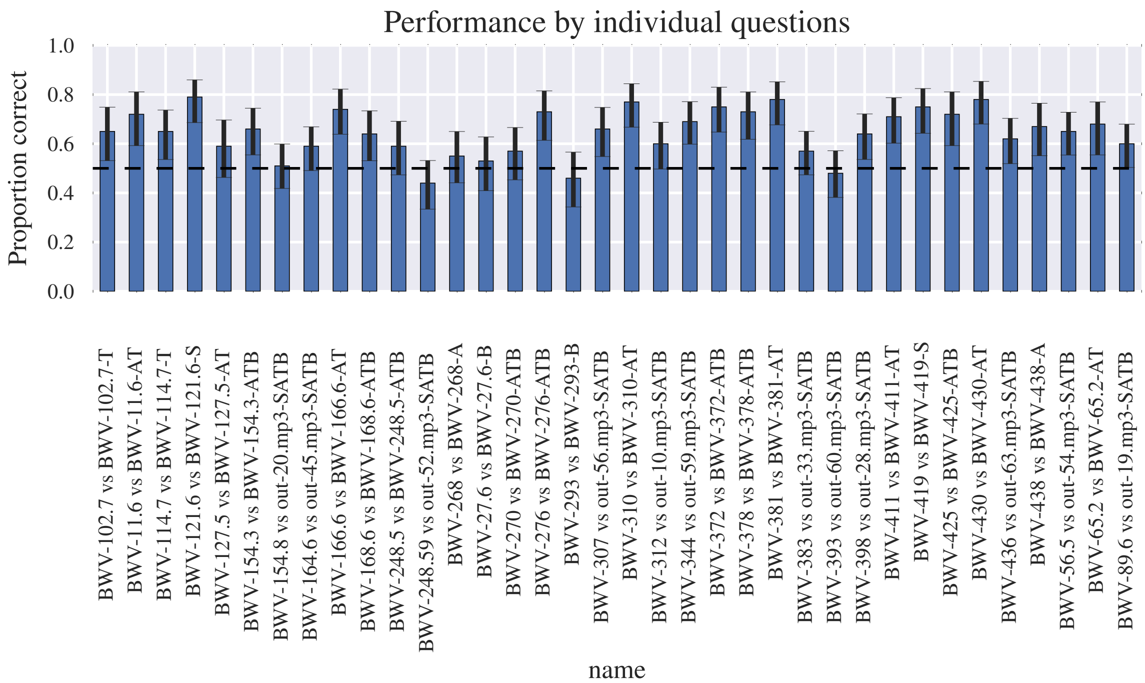


Fig. 6.9 Proportion of correct responses broken down by individual questions.

Paid service providers cost anywhere from $20 to $55 per month just for authoring tools and server space[117] At the time of writing, paid responses cost $1.50–$3.00 on SurveyMonkey [uks].

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 [13]

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.

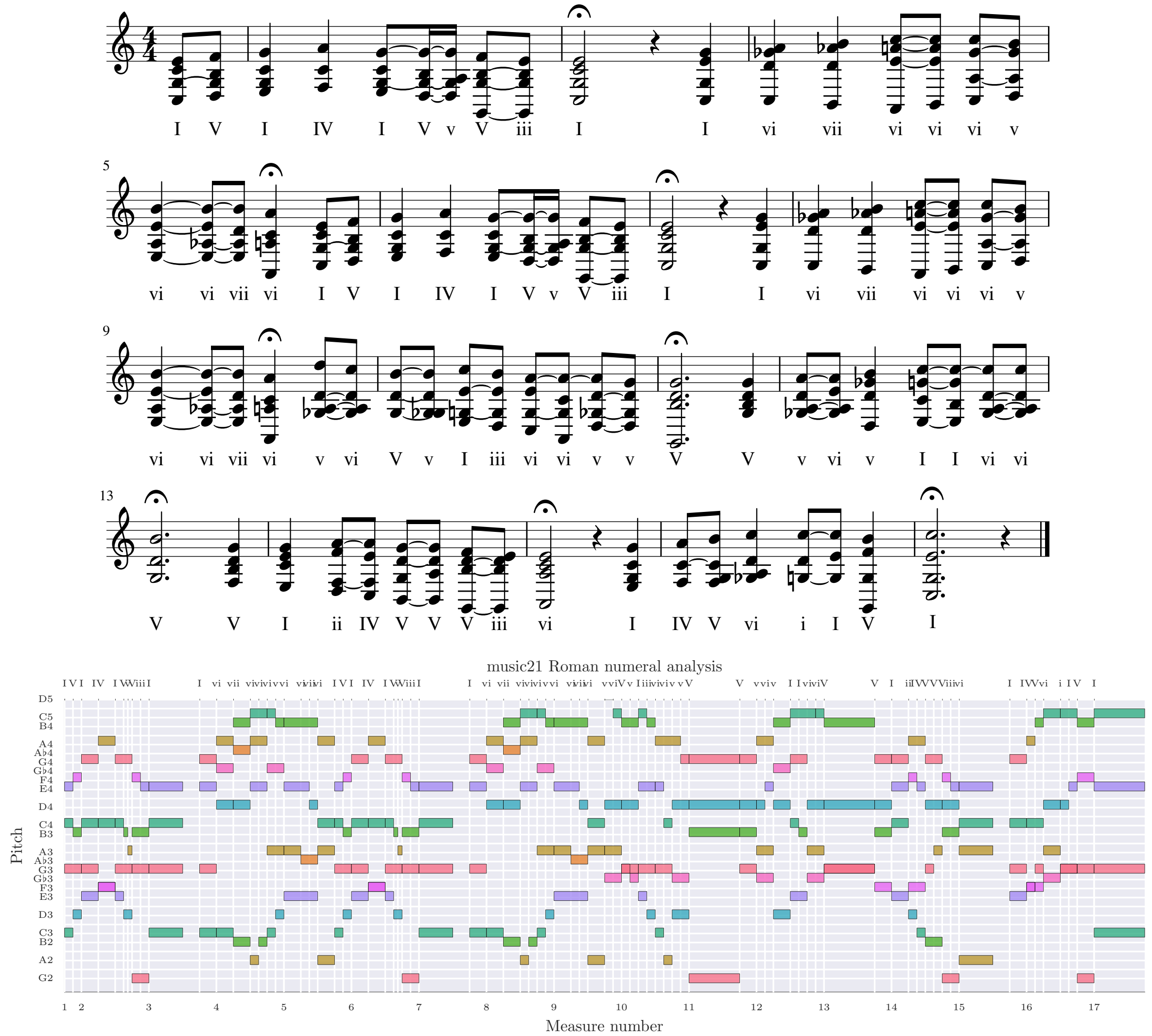


Fig. 7.1 *Top*: The preprocessed score (BWV 133.6) used as input stimulus with Roman numeral analysis annotations obtained from `music21`; *Bottom*: The same stimulus represented on a piano roll

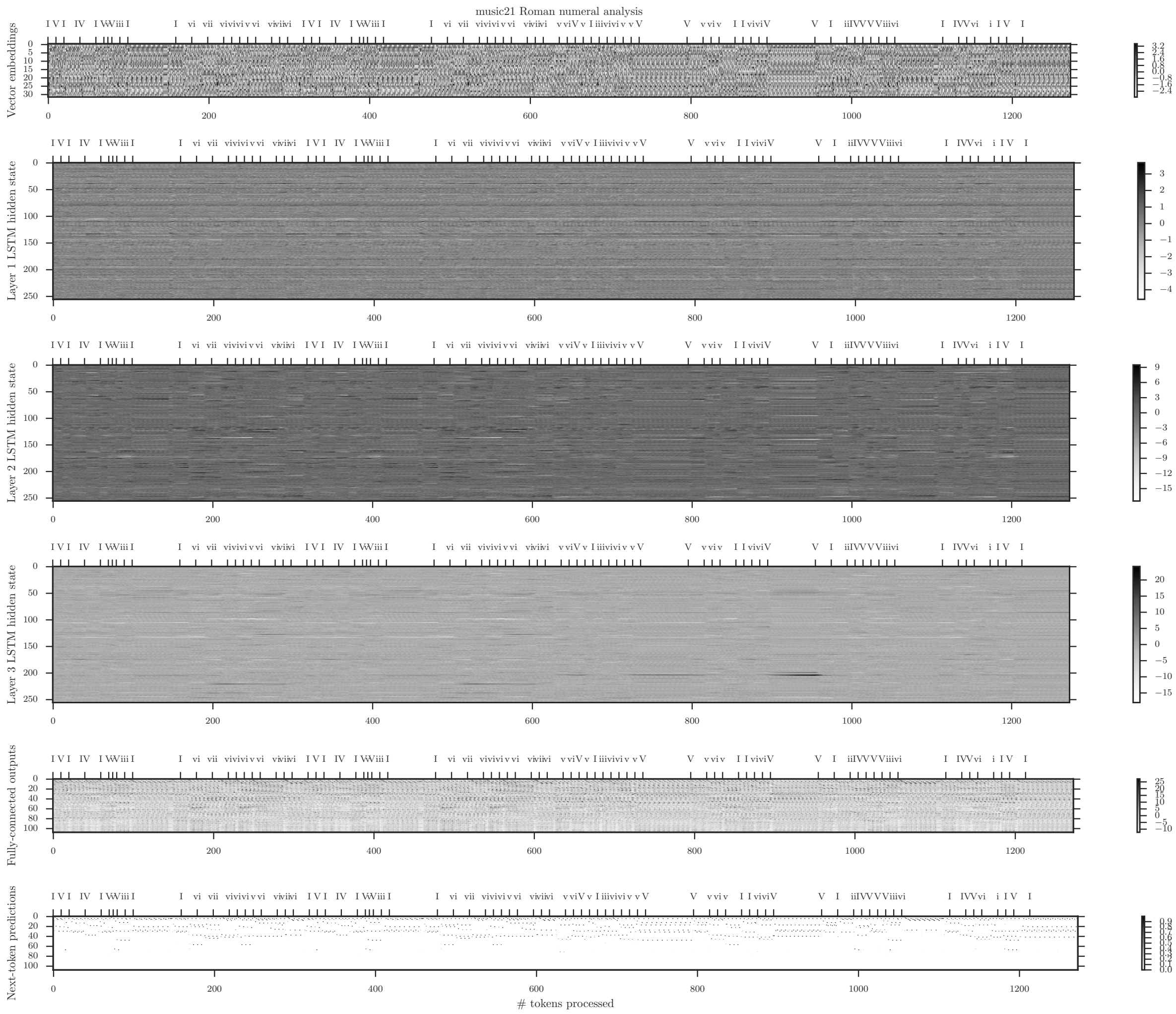


Fig. 7.2 Neuron activations over time as the encoded stimulus is processed token-by-token

7.1.1 Pooling over frames

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

fliang: cite

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

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

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

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

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

The max-pooled frame-level activations are shown in **??** . 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**[32] consisting of the model's sequence of next-frame predictions as it processes the input.

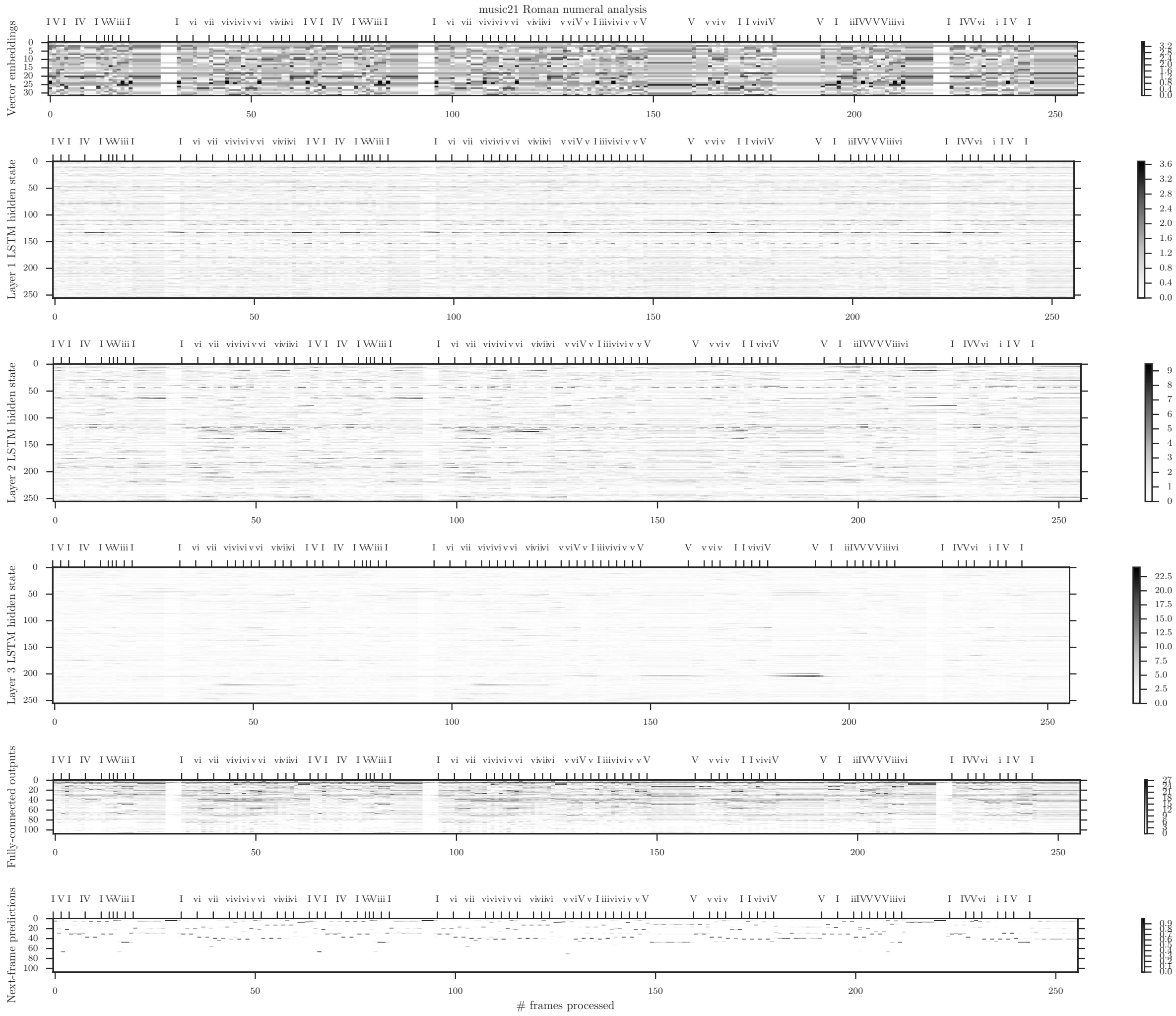


Fig. 7.3 Neuron activations over time pooled over frames

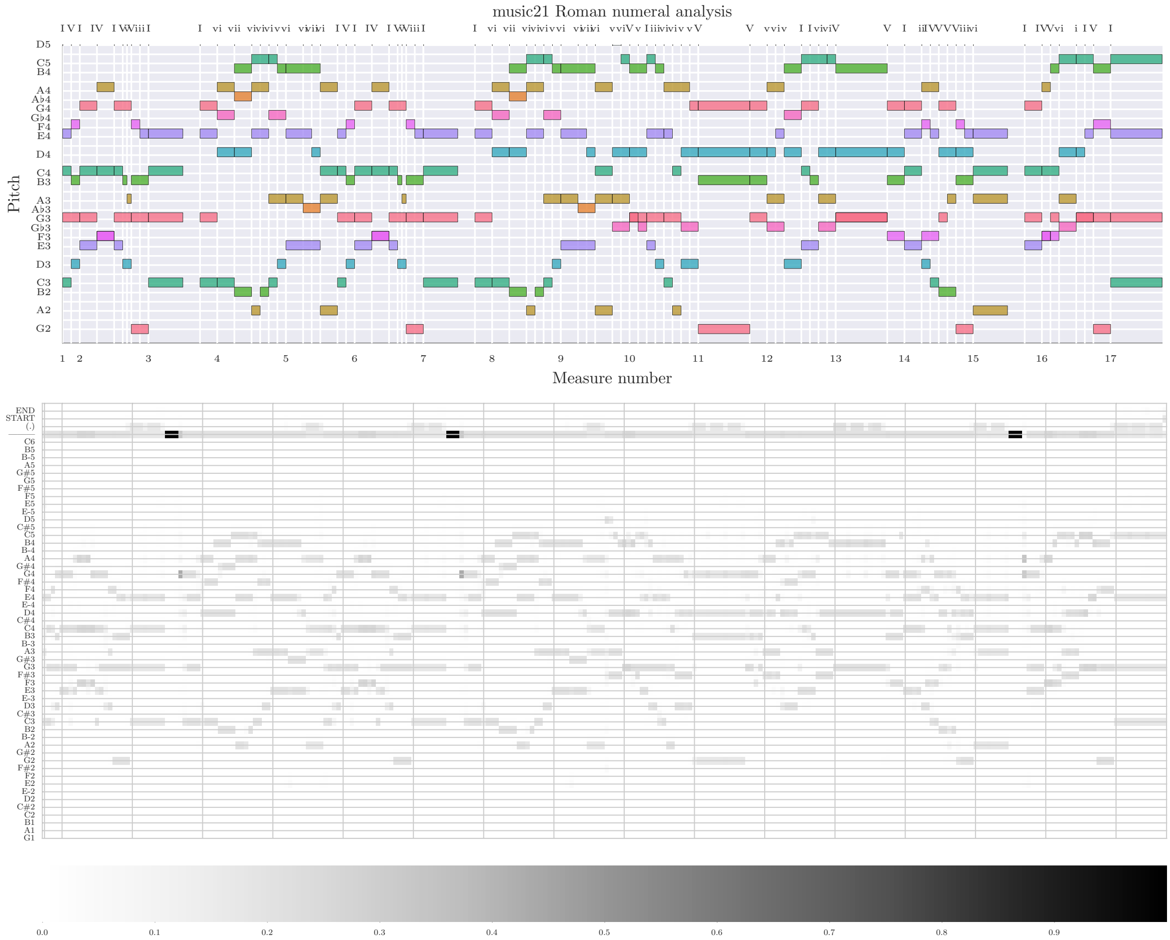


Fig. 7.4 Top: piano roll of stimulus (included for reference); Bottom: probabilistic piano roll

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. [114]), 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 occuring directly after a rest is less constrained than

fliang: cite the intuition?

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

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

7.1.3 Neurons specific to musical concepts

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

fliang: Cite deconvolution

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

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

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

The first three neurons ($(1,64)$, $(1,138)$, $(1,207)$) shown in the 2nd to 4th panel from top of **??** 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 occuring at the $v - -vi$ progressions near frames 55, 120, 130, and a lower intensity spike at 170. Some activity is also observed for the $V - -vi$ around frame 230 despite the first chord being a major mode V rather than minor v.

8

Discussion, Conclusions, and Future Work

8.1 Discussion

8.1.1 Contributions

- An encoding scheme for representing music with arbitrary degrees of polyphony as ordered sequences of tokens
- Brought together recent methods from deep learning to develop a sequence prediction model which avoids any hand-crafted input features and minimizes domain-specific design choices
- Optimized the performance of the proposed model and quantitatively evaluated its performance on both composition and harmonization tasks
- Performed the largest (as of Tuesday 9th August, 2016) published musical Turing test of an automatic composition system
- Investigated the internal representations learned by the model, identifying neurons specific to music-theoretic concepts.

8.2 Conclusions

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

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

8.3 Future work

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

One method to address this is to apply bidirectional RNNs[51] and the sequence to sequence framework[102] to map the constrained parts to the harmonized parts while accounting for both past and future context. An attention mechanism similar to Bahdanau et al. [7] could be introduced on top of the bidirectional RNN to both enable the model to selectively attend to specific time intervals within the context as well as provide insight into what the model deems relevant when generating harmonies.

Another way to account for future constraints is using a lookahead search. Instead of generating outputs by greedily sampling the RNN's predictions at each time, the RNN is expanded for multiple timesteps and the overall best path is selected. This approach is significantly complicated by an exponential growth in possible states ($O(128^L)$ states when looking ahead L timesteps), but nevertheless can be made computationally tractable using approximation techniques like beam search[82].

Another interesting area for further exploration is in how the different parts are ordered when flattening music scores (where all the notes in a chord are played simultaneously) into encoded token sequences (where a sequential ordering is imposed on the notes in a chord). Harmonization results from 5.4 on page 72 showed that the Soprano and Alto parts achieved significantly higher error rates than Tenor and Bass parts, which might be attributed the SATB order imposed when encoding chords. We expect ordering the parts to be harmonized last should improve the model as the fixed parts can now provide additional context aiding more consistent harmony prediction.

We use the Bach chorales because they are fairly homogeneous, widely available in a respectable quantity, and well studied by music theorists. However, our proposed encoding scheme extends to arbitrary degrees of polyphony and musical style. As evidenced by 5.3 on page 74, the learned model is already able to plausibly harmonize melodies which differ significantly from Bach's baroque style. An interesting extension would be to further investigate the limits of of our model's generalality and its failure modes by applying the model to other styles of music.

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

References

[uks] Online research panel pricing | surveymonkey audience.

[2] (2015). abc:standard [abc wiki].

[3] Allan, M. and Williams, C. K. (2005). allan2005. *Advances in Neural Information Processing Systems*, 17:25–32.

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

[5] Ariza, C. (2009). The interrogator as critic: The turing test and the evaluation of generative music systems. *Computer Music Journal*, 33(2):48–70.

[6] 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.

[7] Bahdanau, D., Cho, K., and Bengio, Y. (2015). #4 Neural Machine Translation by Jointly Learning to Align and Translate. pages 1–15.

[8] Bellgard, M. I. and Tsang, C.-P. (1994). Harmonizing music the boltzmann way. *Connection Science*, 6(2-3):281–297.

[9] 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.

[10] Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127.

[11] 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.

[12] 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.

[13] Bharucha, J. J. and Todd, P. M. (1989). Modeling the perception of tonal structure with neural nets. *Computer Music Journal*, 13(4):44–53.

[14] 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.

[15] Bowles, E. (1970). Musicke's handmaiden: Or technology in the service of the arts. *The computer and music*, pages 3–20.

[16] 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.

[17] Butt, J. (1999). Bach-werke-verzeichnis. *Notes*, 55(4):890–893.

[18] 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*.

[19] Cooper, G. and Meyer, L. B. (1963). *The rhythmic structure of music*, volume 118. University of Chicago Press.

[20] Cope, D. (1992). Computer modeling of musical intelligence in emi. *Computer Music Journal*, 16(2):69–83.

[21] 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.

[22] 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.

[23] Cuthbert, M. S. and Ariza, C. (2010). music21: A toolkit for computer-aided musicology and symbolic music data.

[24] 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.

[25] Cybenko, G. (1993). Degree of approximation by superpositions of a sigmoidal function. *Approximation Theory and its Applications*, 9(3):17–28.

[26] Dannenberg, R. B., Thom, B., and Watson, D. (1997). A machine learning approach to musical style recognition.

[27] Denny, J. (1960). *The Oxford school harmony course*, volume 1. Oxford University Press.

[28] Denton, C. and Fillion, M. (1997). The history of musical tuning and temperament during the classical and romantic periods.

[29] Dolson, M. (1989). Machine tongues xii: Neural networks. *Computer Music Journal*, 13(3):28–40.

[30] 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.

[31] Ebcioğlu, K. (1988). An expert system for harmonizing four-part chorales. *Computer Music Journal*, 12(3):43–51.

[32] Eck, D. and Lapalme, J. (2008). Learning musical structure directly from sequences of music. *University of Montreal, Department of Computer Science, CP*, 6128.

[33] Eck, D. and Schmidhuber, J. (2002a). A First Look at Music Composition using LSTM Recurrent Neural Networks. *Idsia*.

[34] 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.

[35] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

[36] 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.

[37] Franklin, J., Dietterich, T., Becker, S., and Ghahramani, Z. (2001). Learning and improvisation. In *Neural Information Processing Systems*, volume 14.

[38] Franklin, J. A. (2004a). Predicting reinforcement of pitch sequences via lstm and td. In *Proc. of International Computer Music Conference, Miami, Florida*, volume 306.

[39] Franklin, J. A. (2004b). Recurrent neural networks and pitch representations for music tasks. In *FLAIRS Conference*, pages 33–37.

[40] Franklin, J. A. (2005). Jazz melody generation from recurrent network learning of several human melodies. In *FLAIRS Conference*, pages 57–62.

[41] Franklin, J. A. (2006). Recurrent neural networks for music computation. *INFORMS Journal on Computing*, 18(3):321–338.

[42] Freedman, D. (2015). *Correlational Harmonic Metrics: Bridging Computational and Human Notions of Musical Harmony*. PhD thesis.

[43] Gers, F. A., Pérez-Ortiz, J. A., Eck, D., and Schmidhuber, J. (2002a). Dekf-lstm. In *ESANN*, pages 369–376.

[44] 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.

[45] 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.

[46] Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471.

[47] 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.

[48] 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.

[49] 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.

[50] Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

[51] Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM networks. *Proceedings of the International Joint Conference on Neural Networks*, 4:2047–2052.

[52] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2015). Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*.

[53] Griffith, N. and Todd, P. M. (1999). *Musical networks: Parallel distributed perception and performance*. MIT Press.

[54] Group, P. R. et al. (1986). Parallel distributed processing: Explorations in the microstructure of cognition: Vol. 1. *Foundations. Cambridge, MA: MIT Press*.

[55] Handel, S. (1993). *Listening: An introduction to the perception of auditory events.* The MIT Press.

[56] Heafield, K., Pouzyrevsky, I., Clark, J. H., and Koehn, P. (2013). Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, Sofia, Bulgaria.

[57] Herlands, W., Der, R., Greenberg, Y., and Levin, S. (2014). A Machine Learning Approach to Musically Meaningful Homogeneous Style Classification. *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 276–282.

[58] Hild, H., Feulner, J., and Menzel, W. (1991). Harmonet: A neural net for harmonizing chorales in the style of js bach. In *NIPS*, pages 267–274.

[59] Hinton, G. E. and Sejnowski, T. J. (1986). Learning and releaming in boltzmann machines. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1:282–317.

[60] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[61] Hoffmann, P. (2002). Towards an" automated art": Algorithmic processes in xenakis' compositions. *Contemporary Music Review*, 21(2-3):121–131.

[62] Hörnel, D. (1997). Melonet i: Neural nets for inventing baroque-style chorale variations. In *NIPS*, pages 887–893.

[63] Hornel, D. and Ragg, T. (1996). Learning musical structure and style by recognition, prediction and evolution.

[64] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[65] Jordan, M. I. (1997). Serial order: A parallel distributed processing approach. *Advances in psychology*, 121:471–495.

[66] Koutnik, J., Greff, K., Gomez, F., and Schmidhuber, J. (2014). A Clockwork RNN. *Proceedings of The 31st International Conference on Machine Learning*, 32:1863–1871.

[67] Krumhansl, C. L. (2001). *Cognitive foundations of musical pitch*. Oxford University Press.

[68] Laden, B. and Keefe, D. H. (1989). The representation of pitch in a neural net model of chord classification. *Computer Music Journal*, 13(4):12–26.

[69] Lerdahl, F. (1983). Jackendoff. a generative theory of tonal music.

[70] 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.

[71] Liu, I.-T. and Ramakrishnan, B. (2014). Bach in 2014: Music Composition with Recurrent Neural Network. *arXiv:1412.3191*, 5:1–9.

[72] 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.

[73] Lyu, Q. (2015). Polyphonic Music Modelling with LSTM-RTRBM. *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pages 991–994.

[74] 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.

[75] Mikolov, T. (2012). *Statistical Language Models Based on Neural Networks*. PhD thesis.

[76] Mikolov, T., Joulin, A., Chopra, S., and Mathieu, M. (2015). Learning Longer Memory in Recurrent Neural Networks. *Iclr*, pages 1–9.

[77] Mikolov, T., Karafiat, M., Burget, L., Cernocky, J., and Khudanpur, S. (2010). Recurrent Neural Network based Language Model. *Interspeech*, (September):1045–1048.

[78] 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.

[79] Nattiez, J.-J. (1990). *Music and discourse: Toward a semiology of music*. Princeton University Press.

[80] 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.

[81] 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.

[82] Norvig, P. (1992). *Paradigms of artificial intelligence programming: case studies in Common LISP*. Morgan Kaufmann.

[83] Oswald, J. (1973). Harmony: Schenkerian analysis.

[84] 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.

[85] 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.

[86] Pearce, M., Meredith, D., and Wiggins, G. (2002). Motivations and methodologies for automation of the compositional process. *Musicae Scientiae*, 6(2):119–147.

[87] Pearce, M. and Wiggins, G. (2001). Towards a framework for the evaluation of machine compositions. In *Proceedings of the AISB'01 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*, pages 22–32. Citeseer.

[88] Piston, W. (1978). Harmony. (revised and expanded by mark devoto). *Londres: Victor Gollancz LTD*.

[89] Ramage, D. (2007). Hidden markov models fundamentals. *Lecture Notes. http://cs229. stanford. edu/section/cs229-hmm. pdf*.

[90] Randel, D. M. (1999). *The Harvard concise dictionary of music and musicians*. Harvard University Press.

[91] Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference On*, pages 586–591. IEEE.

[92] Robinson, A. and Fallside, F. (1987). *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering.

[93] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

[94] Shepard, R. N. (1982). Geometrical approximations to the structure of musical pitch. *Psychological review*, 89(4):305.

[95] Spangler, R. R., Goodman, R. M., and Hawkins, J. (1998). Bach in a box-real-time harmony.

[96] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

[97] Stamatatos, E. and Widmer, G. (2005). Automatic identification of music performers with learning ensembles. *Artificial Intelligence*, 165(1):37–56.

[98] Stolcke, A. et al. (2002). Srilm-an extensible language modeling toolkit. In *Interspeech*, volume 2002, page 2002.

[99] Sturm, B., Santos, J. F., and Korshunova, I. (2015). Folk music style modelling by recurrent neural networks with long short term memory units. In *16th International Society for Music Information Retrieval Conference*.

[100] Sturm, B. L., Santos, J. F., Ben-Tal, O., and Korshunova, I. (2016). Music transcription modelling and composition using deep learning. *arXiv preprint arXiv:1604.08723*.

[Sutskever] Sutskever, I. Training Recurrent Neural Networks - Ilia Sutskever - PhD thesis.

[102] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[103] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

[104] Terhardt, E. (1974). Pitch, consonance, and harmony. *The Journal of the Acoustical Society of America*, 55(5):1061–1069.

[105] Todd, P. (1988). A sequential network design for musical applications. In *Proceedings of the 1988 connectionist models summer school*, pages 76–84.

[106] Todd, P. M. (1989). A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43.

[107] Toiviainen, P. (2000). Symbolic ai versus connectionism in music research.

[108] Tsang, C. P. and Aitken, M. (1991). Harmonizing music as a discipline in contraint logic programming. In *Proceedings of the International Computer Music Conference*, pages 61–61. INTERNATIONAL COMPUTER MUSIC ACCOCIATION.

[109] Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.

[110] Tymoczko, D. (2009). Three conceptions of musical distance. *Communications in Computer and Information Science*, 38:258–272.

[111] Wanner, E. (1980). The atn and the sausage machine: Which one is baloney? *Cognition*, 8(2):209–225.

[112] White, J. D. and Lake, W. E. (2002). *Guidelines for college teaching of music theory*. Scarecrow Press.

[113] 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.

[114] Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

[115] 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.

[116] Winograd, T. (1968). Linguistics and the computer analysis of tonal harmony. *journal of Music Theory*, 12(1):2–49.

[117] 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.

[118] Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

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

1

A

Installing the CUED class file

LaTeX.cls files can be accessed system-wide when they are placed in the <texmf>/tex/latex directory, where <texmf> is the root directory of the user's TeXinstallation. On systems that have a local texmf tree (<texmflocal>), which may be named "texmf-local" or "localtexmf", it may be advisable to install packages in <texmflocal>, rather than <texmf> as the contents of the former, unlike that of the latter, are preserved after the LaTeXsystem is reinstalled and/or upgraded.

It is recommended that the user create a subdirectory <texmf>/tex/latex/CUED for all CUED related LaTeXclass and package files. On some LaTeXsystems, the directory look-up tables will need to be refreshed after making additions or deletions to the system files. For TeXLive systems this is accomplished via executing "texhash" as root. MIKTeXusers 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.

9
Graveyard

9.1 Neural Networks

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

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.

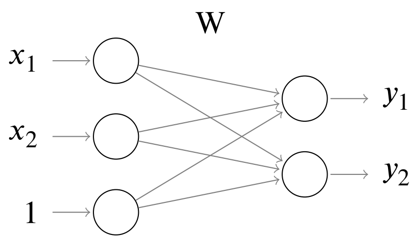


Fig. 9.1 Single feedfoward neural network layer

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

fliang: CITE: fig:ffw-nn

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

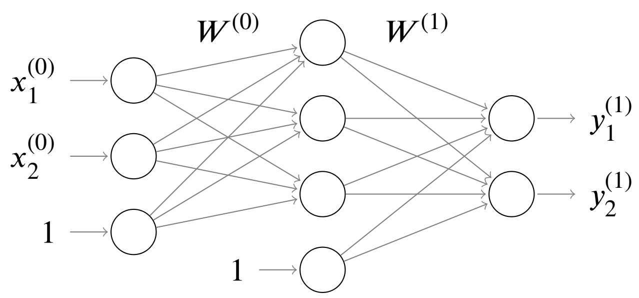


Fig. 9.2 2-layer feedforward neural network

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

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

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

Modeling probability distributions

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

fliang: CITE: eq:softmax

.

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

Efficient graident computations through back-propogation

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

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

fliang: cite theano

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

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

fliang: Cite

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

Vanishing gradients

The solution is to rewrite

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

such that

fliang: CITE: eq:prod-hi

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

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

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

Training with back-propogation

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

fliang: CITE: eq:cross-entropy-loss

):

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

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

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

).

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

Variants of

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

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

fliang: Discuss RMSprop?

To apply

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

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

fliang: cite

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

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

fliang: CITE: eq:cross-entropy-loss

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

For earlier layers $l < L$, we have

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

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

fliang: DO THIS

$$
\circ \tag{9.8}
$$

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

The back-propogation algorithm consists of two steps:

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

2. *Backward pass*: Recursively iterate

   fliang: CITE: eq:backprop

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

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

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

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

Some appealing properties of backpropogation:

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

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

9.2 RNNs

The advantages of RNNs over feedforward networks include:

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

Comparison against HMMs

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

fliang: Define HMMs

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

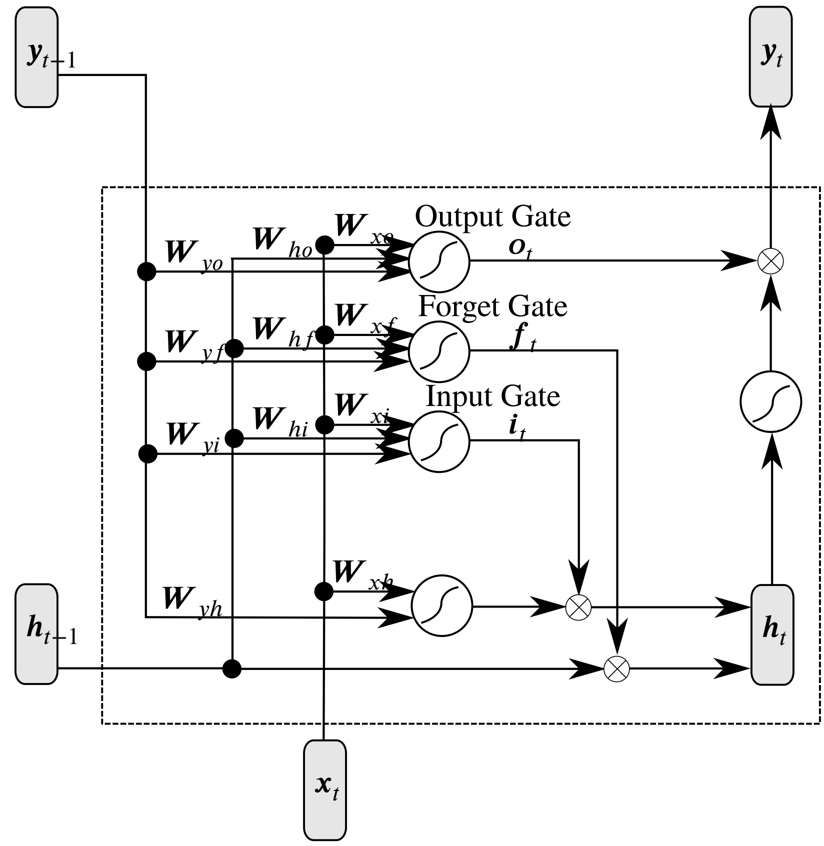


Fig. 9.3 Single LSTM unit

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.

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

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

9.4 Related work

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

The application of machine learning to music has a rich history. [57] describe a system to classify music into homogeneous styles. However, they focus on the discriminative task and do not consider how to generate novel scores.

9.5 LSTMs: background and motivation

Two prominent methods for training RNNs include real-time recurrent learning (RTRL) [92] and backpropogation through time (BPTT) [115]. [113] introduces truncated BPTT to address computational complexity when learning over very long sequences. Temporal difference [103] has also been proposed as a method for learning RNNs [38].

The first LSTM models, which did not include forget gates, was introduced in [60]. [46] later revised the LSTM model to include forget gates in order to prevent hidden states from growing indefinitely.

LSTMs have been demonstrated to outperform traditional RNNs on a variety of tasks. [44] demonstrates a LSTM correctly recognizing 1000 instances from the context-free grammar $A^n B^n$ while an Elman RNN achieves only 20% accuracy.

Online adaptation at test time using a Kalman filter was described in [43]. [77] [75] refers to this as “dynamic evaluation.”

In *Bach in a Box* [95], harmonic rules are collected in a database and then used to build rule-based neural networks. This enables encoding of prior knowledge as rules in the rulebase.

[32] attempts to model meter by introducing time-delayed connections in [33]

9.5.1 Representation of music data

[78] discusses the importance of music representation, settling on *psychologically-based representations* of pitch, duration, and harmonic structure [94].

Many attempts to represent musical data have been investigated. Attempts which explicitly model harmonic structure include a Circle of Thirds representation [39] or overlapping subharmonics representation[68], both of which have been studied in the context of generative RNN models [39] [78]. Other representations attempt to model notions such as musical distance in terms of voice leading, orbifolds, and tuning lattices[110].

[40] 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

[87] addresses difficulty in quantitative evaluation, suggesting the use of a learned critic in a manner similar to GANs [49]. In a later report, [86] attribute difficulty in evaluation due to lack aim: algorithmic composition, design of compositional tools, and computational modelling of musical styles or music cognition all have different motivations and should thus be evaluated differently.

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

9.6 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:

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

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.

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

fliang: cite gird LSTMs

.

9.7 Token-level embeddings

fliang: EXPERIMENT: redo these

Filter to notes

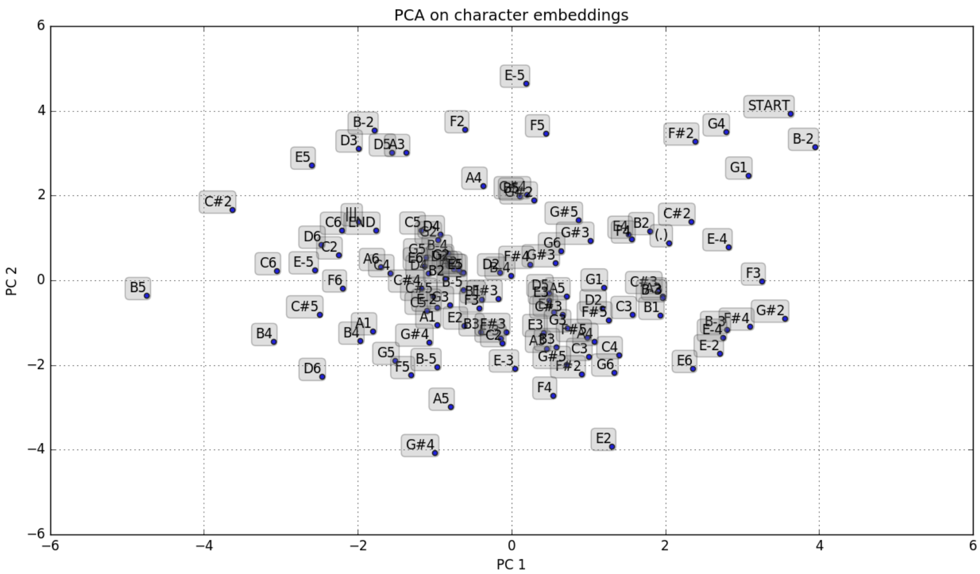


Fig. 9.4 PCA embedding of note tokens

9.7.1 Variable-length embeddings

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

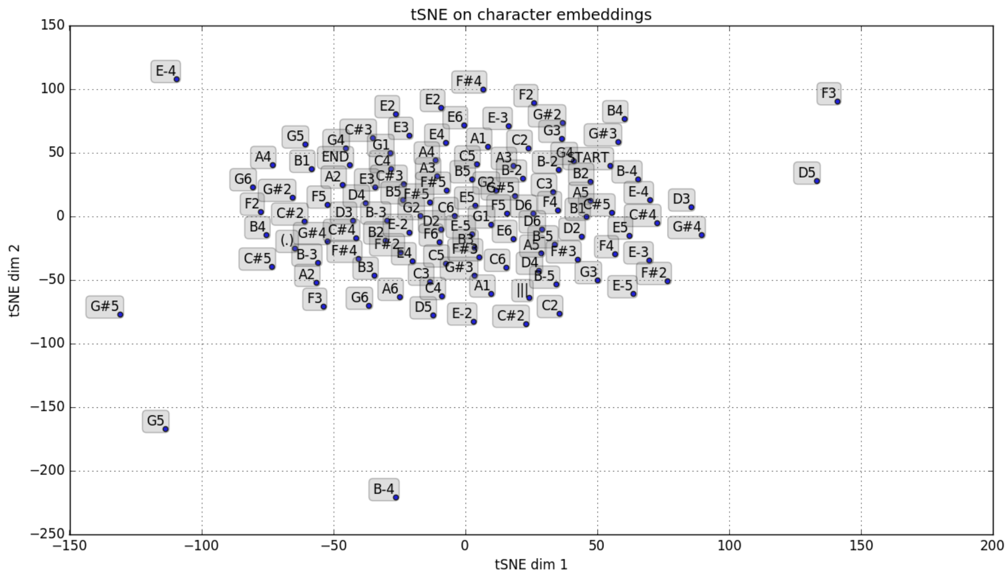


Fig. 9.5 tSNE embedding of note tokens