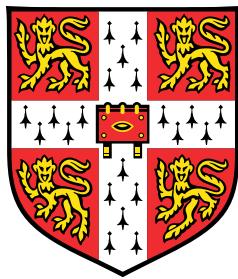


# BachBot: A generative model for Bach chorales

Automatic stylistic composition with deep long short-term memory



**Feynman Liang**

Department of Engineering  
University of Cambridge

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

This dissertation is submitted for the degree of  
*Masters of Philosophy*

Churchill College

August 2016

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

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

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

## 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

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

## Acknowledgements

And I would like to acknowledge ...

fliang: Make everyone feel appreciated

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

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

**Bill Byrne** : setting timeline, general guidance, proofreading

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

**Christopher Hicks** : providing expert music feedback

**Anna Langley** : providing user feedback and promoting within CUED

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

**Marcin Tomczak** : for discussions

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

## Abstract

This is where you write your abstract ...

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

# Table of contents

<b>List of figures</b>	<b>15</b>
<b>List of tables</b>	<b>19</b>
<b>1 Introduction</b>	<b>21</b>
<b>2 Background</b>	<b>25</b>
2.1 Recurrent neural networks . . . . .	25
2.1.1 Notation . . . . .	25
2.1.2 The memory cell abstraction . . . . .	26
2.1.3 Operations on RNNs: unrolling and stacking . . . . .	27
2.1.4 Training RNNs and backpropagation through time . . . . .	28
2.1.5 Long short term memory: solving the vanishing gradient . . . . .	31
2.2 Sequence probability modelling . . . . .	33
<b>3 Related Work</b>	<b>37</b>
3.1 Machine learning on musical data . . . . .	37
3.2 Models for automatic composition . . . . .	38
3.2.1 Symbolic rule-based methods . . . . .	38
3.2.2 Connectionist methods . . . . .	39
3.2.3 Hybrid methods . . . . .	40
3.2.4 LSTM music synthesis models . . . . .	41
3.3 Generative modelling of Bach Chorales . . . . .	42
<b>4 Automatic composition with deep LSTM</b>	<b>45</b>
4.1 Constructing a corpus of encoded Bach chorales scores . . . . .	45
4.1.1 Preprocessing . . . . .	46
4.1.2 Sequential encoding of musical data . . . . .	51

4.2	Design and validation of a generative model for music . . . . .	54
4.2.1	Training and evaluation criteria . . . . .	54
4.2.2	Establishing a baseline with $N$ -gram language models . . . . .	55
4.2.3	Description of RNN model hyperparameters . . . . .	55
4.2.4	Comparison of memory cells on music data . . . . .	57
4.2.5	Optimizing the LSTM architecture . . . . .	57
4.3	Results . . . . .	60
4.4	Other applications . . . . .	61
<b>5</b>	<b>Analysis of musical concepts learned by the model</b>	<b>63</b>
5.1	Investigation of neuron activation responses to applied stimulus . . . . .	63
5.1.1	Pooling over frames . . . . .	64
5.1.2	Probabilistic piano roll: likely variations of the stimulus . . . . .	64
5.1.3	Neurons specific to musical concepts . . . . .	65
<b>6</b>	<b>Chorale harmonization</b>	<b>73</b>
6.1	Background . . . . .	73
6.2	Harmonizing . . . . .	74
6.3	Datasets . . . . .	76
6.4	Results . . . . .	76
6.4.1	Harmonizing popular tunes with BachBot . . . . .	77
<b>7</b>	<b>Large-scale subjective evaluation</b>	<b>79</b>
7.1	Evaluation framework design . . . . .	80
7.1.1	Software architecture . . . . .	80
7.1.2	User interface . . . . .	80
7.1.3	Question generation . . . . .	81
7.1.4	Promoting the study . . . . .	81
7.2	Results . . . . .	82
7.2.1	Participant backgrounds and demographics . . . . .	82
7.2.2	BachBot's performance results . . . . .	83
7.3	User feedback . . . . .	86
7.4	Competitive analysis of large-scale evaluation methodologies . . . . .	89
<b>8</b>	<b>Discussion, Conclusions, and Future Work</b>	<b>91</b>
8.1	Discussion . . . . .	91
8.1.1	Contributions . . . . .	91

---

Table of contents	<b>13</b>
8.2 Conclusions . . . . .	92
8.3 Extensions and Future Work . . . . .	92
<b>References</b>	<b>95</b>
<b>Appendix A Appendix A</b>	<b>105</b>
A.1 A primer on Western music theory . . . . .	105
A.1.1 Notes: the basic building blocks . . . . .	106
A.1.2 Tonality in common practice music . . . . .	109
A.1.3 Polyphony, chords, and chord progressions . . . . .	110
A.2 An introduction to neural networks . . . . .	112
A.2.1 Neurons: the basic computation unit . . . . .	112
A.2.2 Feedforward neural networks . . . . .	113
A.2.3 Recurrent neural networks . . . . .	114
A.3 Sufficient conditions for vanishing gradients . . . . .	115
<b>Appendix B Appendix B</b>	<b>117</b>
B.1 Additional Figures and Tables . . . . .	117
<b>9 Graveyard</b>	<b>127</b>
9.1 Neural Networks . . . . .	127
9.2 RNNs . . . . .	132
9.3 Sequence probability modelling . . . . .	133
9.4 Related work . . . . .	134
9.5 LSTM: background and motivation . . . . .	134
9.5.1 Representation of music data . . . . .	134
9.5.2 Evaluation of models . . . . .	135
9.6 Automatic Composition . . . . .	135
9.6.1 Multi-GPU implementation . . . . .	135
9.7 Token-level embeddings . . . . .	136
9.7.1 Variable-length embeddings . . . . .	136

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

# List of figures

2.1	An Elman-type RNN with a single hidden layer. The recurrent hidden state is illustrated as unit-delayed (denoted by $z^{-1}$ ) feedback edges from the hidden states to the input layer. The memory cell encapsulating the hidden state is also shown. . . . .	27
2.2	Signal flow diagram representation of a single-layer RNN (left) and its corresponding DAG (right) after unrolling. The blocks labelled with $\mathbf{h}_t$ represent memory cells whose parameters are shared across all times $t$ . . . . .	28
2.3	Block diagram representation of a -layer RNN (left) and its corresponding DAG (right) after unrolling. The blocks labelled with $\mathbf{h}_t$ represent memory cells whose parameters are shared across all times $t$ . . . . .	29
2.4	The gradients passed along network edges during BPTT. . . . .	30
2.5	Schematic for a single LSTM memory cell. Notice how the gates $i_t$ , $o_t$ , and $f_t$ control access to the constant error carousel (CEC). . . . .	32
4.1	First 4 bars of JCB Chorale BWV 185.6 before (top) and after (bottom) pre-processing. Note the transposition down by a semitone to C-major as well as quantization of the demisemiquavers in the third bar of the Soprano part. . . .	47
4.2	Piano roll representation of the same 4 bars from fig. 4.1 before and after pre-processing. Again, note the transposition to C-major and time-quantization occurring in the Soprano part. . . . .	49
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. . . . .	50
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. . . .	51
4.5	Distribution of note durations over Bach chorales corpus. Quantization has minimal impact because of the high resolution (semiquavers) used. . . . .	51

4.6	Meter is minimally affected by quantization due to the high resolution used for time quantization. . . . .	52
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. . . . .	53
4.8	LSTM and GRUs yield the lowest training loss. Validation loss traces show all architectures exhibit signs of significant overfitting . . . . .	57
4.9	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. . . . .	58
4.10	Training curves for the overall best model. The periodic spikes correspond to resetting of the LSTM state at the end of a training epoch. . . . .	59
5.1	<i>Top:</i> The preprocessed score (BWV 133.6) used as input stimulus with Roman numeral analysis annotations obtained from music21; <i>Bottom:</i> The same stimulus represented on a piano roll . . . . .	67
5.2	Neuron activations over time as the encoded stimulus is processed token-by-token	68
5.3	Neuron activations over time pooled over frames . . . . .	69
5.4	Top: piano roll of stimulus (included for reference); Bottom: probabilistic piano roll . . . . .	70
5.5	Activation profiles of neurons within our model which have learned high-level musical concepts . . . . .	71
6.1	Error rates for harmonization tasks . . . . .	76
6.2	Twinkle-twinkle soprano melody, ATB harmonized by BachBot . . . . .	77
7.1	The first page seen by a visitor of <a href="http://bachbot.com">http://bachbot.com</a> . . . . .	80
7.2	User information form presented after clicking "Test Yourself" . . . . .	81
7.3	Question response interface used for all questions . . . . .	82
7.4	Geographic distribution of participants . . . . .	84
7.5	Demographics of participants . . . . .	85
7.6	responses-Mask . . . . .	85
7.7	Proportion of correct responses for each question type and music experience level. . . . .	86
7.8	Proportion of correct responses for each question type and age group. . . . .	87
7.9	Proportion of correct responses broken down by individual questions. . . . .	88
A.1	Sheet music representation of the first four bars of BWV 133.6 . . . . .	106

A.2	Terhardt's visual analogy for pitch. Similar to how the viewer of this figure may percieve contours not present, pitch describes subjective information received by the listener even when physical frequencies are absent. . . . .	106
A.3	Illustration of an octave in the 12-note chromatic scale on a piano keyboard. .	107
A.4	Scientific pitch notation and sheet music notation of $C$ notes at ten different octaves. . . . .	108
A.5	Comparison of various note durations [23] . . . . .	108
A.6	Piano roll notation of the music in fig. A.1 . . . . .	110
A.7	A single neuron first computes an activation $z$ and then passes it through an activation function $\sigma(\cdot)$ . . . . .	113
A.8	Graph depiction of a feedforward neural network with 2 hidden layers . . . .	114
A.9	Graph representation of an Elman-type RNN. . . . .	115
B.1	Results of grid search (see Section 4.2.5) over LSTM sequence model hyperparameters . . . . .	117
B.2	<code>rnn_size=256</code> and <code>num_layers=3</code> yields lowest validation loss. . . . .	123
B.3	Validation loss improves initially with increasing network depth but deteriorates after $> 3$ layers. . . . .	123
B.4	Validation loss improves initially with higher-dimensional hidden states but deteriorates after $> 256$ dimensions. . . . .	124
B.5	<code>seq_length=128</code> and <code>wordvec=32</code> yields lowest validation loss. . . . .	124
B.6	Perturbations about <code>wordvec=32</code> do not yield significant improvements. . . .	125
9.1	Single feedfoward neural network layer . . . . .	127
9.2	2-layer feedforward neural network . . . . .	128
9.3	Single LSTM unit . . . . .	133
9.4	PCA embedding of note tokens . . . . .	136
9.5	tSNE embedding of note tokens . . . . .	137

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

## List of tables

4.1	Statistics on the preprocessed datasets used throughout our study . . . . .	53
4.2	Perplexities of baseline $N$ -gram language models on encoded music data . .	56
4.3	Timing results comparing CPU and GPU training of the overall best model (section 4.2.5 on page 58) . . . . .	59
7.1	Composition of questions on <a href="http://bachbot.com">http://bachbot.com</a> . . . . .	82
A.1	Pitch intervals for the two most important keys [53]. The pitches in a scale can be found by starting at the tonic and successively offsetting by the given pitch intervals. . . . .	110
A.2	Common chord qualities and their corresponding intervals [53] . . . . .	111

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

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

Alpern [3]

# 1

1

## Introduction

2

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

3

4

5

6

7

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

8

9

10

11

Our contributions include:

12

1. A note-by-note sequential representation for polyphonic music amenable to processing with standard sequence models
- 13
- 14
2. A rigorous investigation of how recent deep learning advances such as dropout [120], batch normalization [77], and new RNN architectures can be applied to improve probabilistic modelling of music data
- 15
- 16
- 17

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

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

5       fliang: XXX

6       participants

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

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

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

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

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

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

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

fliang: Mark: this speaks to an important part of the why for this project and could do with setting in relief.

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

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

# 2

## Background

The goal of this chapter is to provide a selective background in recurrent neural networks and generative probabilistic sequence modelling required for understanding our experiments and results. It also introduces some common definitions and notation used throughout later chapters. We assume a basic understanding of Western music theory. Readers unfamiliar with concepts such as piano rolls, Roman numeral analysis, and cadences, should consult [section A.1](#) on page [105](#) for a quick primer and Piston [107] and Denny [36] for more thorough coverage.

### 2.1 Recurrent neural networks

In this section, we clarify terminology and provide important background in recurrent neural networks (RNNs). While a basic understanding of neural networks is assumed, readers lacking such a background may be interested in reviewing [section A.2](#) on page [112](#) before proceeding.

#### 2.1.1 Notation

We first clarify the basic notation and conventions used to describe multi-layer RNNs. Unless otherwise specified, notation appearing in the remainder of our work is to be interpreted as defined in this section.

<sup>1</sup> We use  $\mathbf{x}_t \in \mathbb{R}^{N_{in}}$  with  $t = 1, 2, \dots, T$  to denote a sequence of **input** (*i.e.* observed) vectors,  
<sup>2</sup>  $\mathbf{z}_t \in \mathbb{R}^{N_{hid}}$  a sequence of **hidden state** (*i.e.* unobserved) vectors. and  $\mathbf{y}_t \in \mathbb{R}^{N_{out}}$  a sequence  
<sup>3</sup> of **output** vectors.

<sup>4</sup> To describe model parameters, we use  $\mathbf{W}$  to indicate the **weight matrix** consisting of all the  
<sup>5</sup> connection weights between two blocks of neurons and  $\sigma(\cdot)$  to indicate the **activation function**.  
<sup>6</sup> The collection of all model parameters is denoted by  $\theta$ .

<sup>7</sup> When further clarity is required, we use  $\mathbf{W}_{s,t}$  to denote the connection weights from block  
<sup>8</sup>  $s$  to block  $t$  (*i.e.* in [section 2.1.5](#),  $\mathbf{W}_{xf}$  and  $\mathbf{W}_{xh}$  refer to the connections from the inputs to the  
<sup>9</sup> forget gate and hidden state respectively). Subscripts on activation functions  $\sigma_{s,t}(\cdot)$  are to be  
<sup>10</sup> interpreted analogously.

<sup>11</sup> Using the above notation, the equations for **RNN time dynamics** can be expressed as

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

<sup>13</sup> When discussing multi-layer networks, we use parenthesized superscripts to indicate layer.  
<sup>14</sup> For example,  $\mathbf{z}_t^{(2)}$  is the hidden states of the second layer and  $N_{in}^{(3)}$  is the dimensionality of the  
<sup>15</sup> third layer's inputs  $\mathbf{x}_t^{(3)}$ . We assume the outputs of the  $l - 1$ st layer are used as the inputs of  
<sup>16</sup> the  $l$ th layer (*i.e.*  $\forall t : \mathbf{x}_t^{(l)} = \mathbf{y}_t^{(l-1)}$ ).

## <sup>17</sup> 2.1.2 The memory cell abstraction

<sup>18</sup> While many different variants for RNNs exist [[46](#), [79](#), [72](#), [20](#), [83](#), [94](#)], many share the same  
<sup>19</sup> underlying structure. Hence, it is useful to discuss RNNs abstractly without specifying a par-  
<sup>20</sup> ticular variant.

<sup>21</sup> To do so, we introduce the notion of a **memory cell** in order to abstract away how different  
<sup>22</sup> variations of RNNs compute  $\mathbf{y}_t$  and  $\mathbf{h}_t$  from  $\mathbf{x}_t$  and  $\mathbf{h}_{t-1}$ . This is illustrated visually in [fig. 2.1](#),  
<sup>23</sup> which shows a standard Elman-type RNN [[46](#)] with the memory cell indicated as a dashed box  
<sup>24</sup> isolating the recurrent hidden state.

<sup>25</sup> Notice that the edges in [fig. 2.1](#) entering the memory cell consist of the input  $\mathbf{x}_t$  and previous  
<sup>26</sup> hidden state  $\mathbf{h}_{t-1}$ , and the edges leaving the memory cell consist of the current hidden state  $\mathbf{h}_t$   
<sup>27</sup> and the outputs  $\mathbf{y}_t$ . Hence, to specify an concrete implementation for a memory cell, it suffices  
<sup>28</sup> to provide two functions  $f_h$  and  $f_y$  which uses  $\mathbf{x}_t$  and  $\mathbf{h}_{t-1}$  to compute the next hidden state  
<sup>29</sup>  $\mathbf{h}_t = f_h(\mathbf{x}_t, \mathbf{h}_{t-1})$  and output  $\mathbf{y}_t = f_y(\mathbf{h}_t)$ .

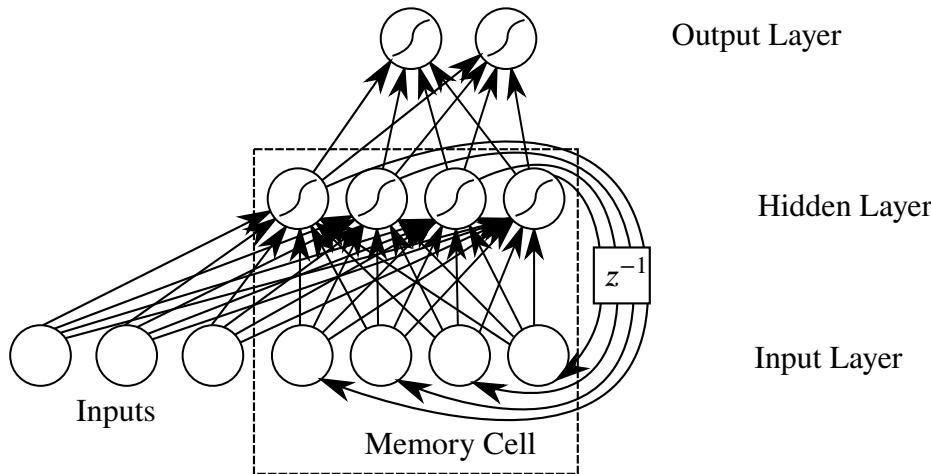


Fig. 2.1 An Elman-type RNN with a single hidden layer. The recurrent hidden state is illustrated as unit-delayed (denoted by  $z^{-1}$ ) feedback edges from the hidden states to the input layer. The memory cell encapsulating the hidden state is also shown.

### 2.1.3 Operations on RNNs: unrolling and stacking

#### Unrolling RNNs into directed acyclic graphs

Given an input sequence  $\{\mathbf{x}_t\}_{t=1}^T$  with length  $T$ , an RNN can be **unrolled** into a directed acyclic graph (DAG) comprised of  $T$  copies of the memory cell connected forwards in time. Figure 2.2 shows a block diagram of fig. 2.1 on the left and its corresponding unrolled DAG on the right.

Figure 2.2 shows that the hidden state  $\mathbf{h}_t$  is passed forwards throughout the sequence of computations. This gives rise to an alternative interpretation of the hidden state as a temporal memory mechanism. Under this interpretation, updating the hidden state  $\mathbf{h}_t = f_h(\mathbf{x}_t, \mathbf{h}_{t-1})$  can be viewed as **writing** information from the current inputs  $\mathbf{x}_t$  to memory and producing the outputs  $\mathbf{y}_t = f_y(\mathbf{h}_t)$  can be interpreted as **reading** information from memory.

#### Stacking memory cells to form deep RNNs

In addition to unrolling, RNNs can also be **stacked** to form deep RNNs [45, 116]. This is accomplished in a manner analogous to deep belief networks: outputs from the previous memory cell in the stack are used as inputs for the current memory cell (see fig. 2.3).

Prior work has observed that “deep RNNs outperformed the conventional, shallow RNN” Pascanu et al. [104], affirming the importance of stacking multiple layers in RNNs. The greater modelling capabilities of multi-layer RNNs can be attributed to three primary factors: composition of multiple non-linear activation functions and an increase in the number of paths for information to flow. The former reason is analogous to the case in deep belief networks, which is well documented [11]. To understand the latter, notice that in fig. 2.2 there is only a single

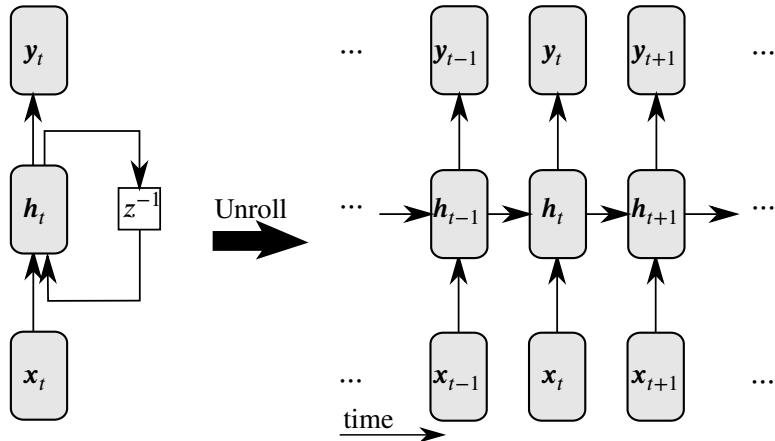


Fig. 2.2 Signal flow diagram representation of a single-layer RNN (left) and its corresponding DAG (right) after unrolling. The blocks labelled with  $h_t$  represent memory cells whose parameters are shared across all times  $t$ .

- <sup>1</sup> path from  $x_{t-1}$  to  $y_t$  hence the conditional independence  $y_t \perp\!\!\!\perp x_{t-1} | h_t^{(1)}$  is satisfied. However,
- <sup>2</sup> in fig. 2.3 there are multiple paths from  $x_{t-1}$  to  $y_t$  (e.g. passing through either  $h_{t-1}^{(2)} \rightarrow h_t^{(2)}$  or
- <sup>3</sup>  $h_{t-1}^{(1)} \rightarrow h_t^{(1)}$ ) through which information may flow.

#### 2.1.4 Training RNNs and backpropagation through time

- <sup>5</sup> The parameters  $\theta$  of a RNN are typically learned from data to minimize some **cost**  $\mathcal{E} =$
- <sup>6</sup>  $\sum_{1 \leq t \leq T} \mathcal{E}_t(x_t)$  measuring the performance of the network on some task. This optimization
- <sup>7</sup> is commonly carried out using iterative gradient descent methods, which require computation
- <sup>8</sup> of the gradients  $\frac{\partial \mathcal{E}}{\partial \theta}$  at each iteration.

<sup>9</sup> In feed-forward networks, computation of gradients can be performed efficiently using  
<sup>10</sup> backpropagation [18, 88, 114]. While the cycles introduced by time-delayed recurrent hid-  
<sup>11</sup> den state connections may seem to complicate matters for RNNs, recall that unrolling removes  
<sup>12</sup> the time-delayed recurrent edges and converts the RNN into a DAG (e.g. fig. 2.2). The unrolled  
<sup>13</sup> RNN can be justifiably interpreted as a  $T$  layered feed-forward neural network with parameters  
<sup>14</sup> shared across all layers, motivating the application of techniques such as backpropagation to  
<sup>15</sup> the unrolled RNNs.

<sup>16</sup> Unsurprisingly, this is precisely what is done in the **backpropagation through time** (BPTT)■  
<sup>17</sup> algorithm [60]. Applying the chain rule to the RNN dynamics equations (eq. (2.1) on page 26)

## 2.1 Recurrent neural networks

## 29

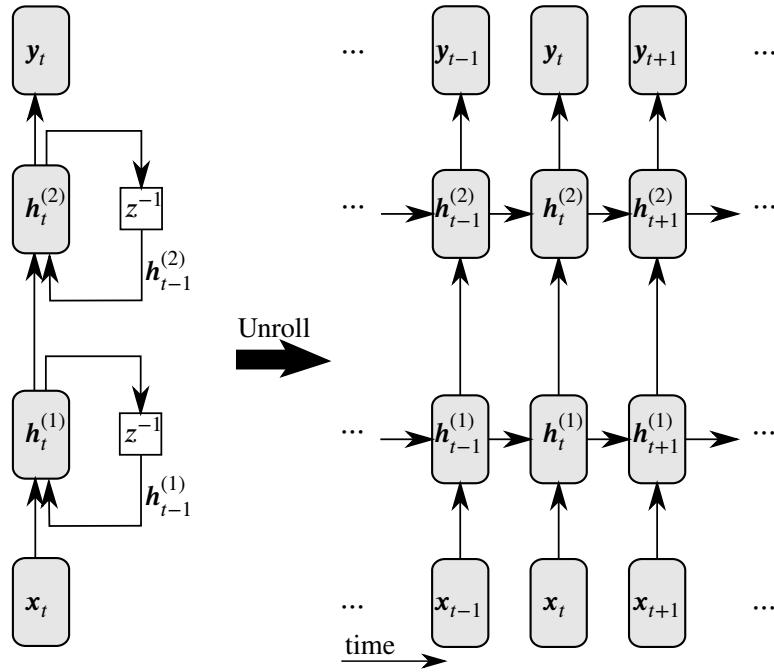


Fig. 2.3 Block diagram representation of a -layer RNN (left) and its corresponding DAG (right) after unrolling. The blocks labelled with  $\mathbf{h}_t$  represent memory cells whose parameters are shared across all times  $t$ .

unrolled network (see fig. 2.4), we obtain

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{E}_t}{\partial \theta} \quad (2.2)$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left( \frac{\partial \mathcal{E}_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \theta} \right) \quad (2.3)$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{t \geq i > k} \mathbf{W}_{hh}^\top \text{diag}(\sigma'_{hh}(\mathbf{h}_{i-1})) \quad (2.4)$$

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

Just like traditional backpropagation, fig. 2.4 demonstrates how BPTT divides the computation of a global gradient  $\frac{\partial \mathcal{E}}{\partial \theta}$  into a series of local gradient computations, each of which involves significantly less variables and hence is easier to compute.

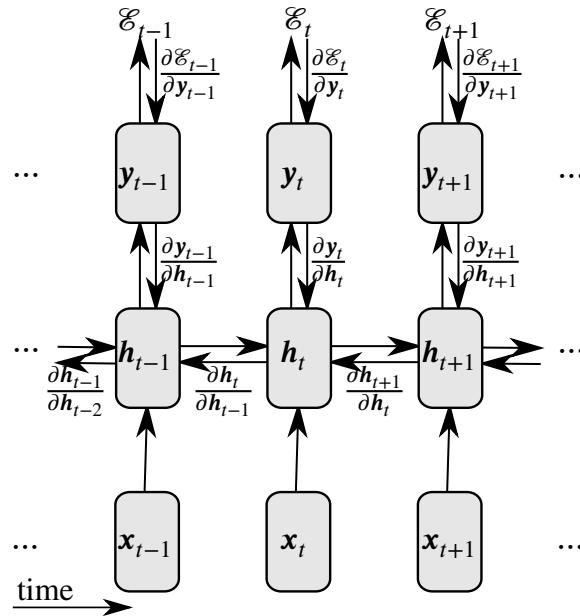


Fig. 2.4 The gradients passed along network edges during BPTT.

## <sup>1</sup> Vanishing/exploding gradients

<sup>2</sup> Naive implementations of RNNs (specifically [eq. \(2.1\)](#)) often suffer from two well known prob-  
<sup>3</sup> lems: the **vanishing gradient** and **exploding gradient** [13]. These problems are both related  
<sup>4</sup> to the product in [eq. \(2.4\)](#) exponentially growing or shrinking over long time-spans (*i.e.*  $t \gg k$ ).  
<sup>5</sup> A sufficient condition (proved in [section A.3](#) on page 115) for vanishing gradients is

$$\| \mathbf{W}_{hh} \| < \frac{1}{\gamma_\sigma} \quad (2.5)$$

<sup>7</sup> where  $\| \cdot \|$  is the matrix operator norm (see [eq. \(A.6\)](#) on page 116),  $\mathbf{W}_{hh}$  is defined in [eq. \(2.1\)](#)  
<sup>8</sup> on page 26, and  $\gamma_\sigma$  is a constant depending on the choice of activation function (*e.g.*  $\gamma_\sigma = 1$  for  
<sup>9</sup>  $\sigma_{hh} = \tanh$ ,  $\gamma_\sigma = 0.25$  for  $\sigma_{hh} = \text{sigmoid}$ ).

<sup>10</sup> This difficulty learning relationships between events spaced far apart in time presents a  
<sup>11</sup> significant challenge for music applications. As noted by Cooper and Meyer [25]:

<sup>12</sup> Long-term dependencies are at the heart of what defines a style of music, with  
<sup>13</sup> events spanning several notes or bars contributing to the formation of metrical and  
<sup>14</sup> phrasal structure.

### 2.1.5 Long short term memory: solving the vanishing gradient

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

$$\mathbf{W}_{hh}^\top \sigma'_{hh}(\mathbf{h}_t) = \mathbf{I} \quad (2.6)$$

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

As a result of the constant error flow condition, notice that eq. (2.4) on page 29 becomes

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{t \geq i > k} \mathbf{W}_{hh}^\top \text{diag}(\sigma'_{hh}(\mathbf{h}_{i-1})) = \prod_{t \geq i > k} \mathbf{I} = \mathbf{I} \quad (2.7)$$

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

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

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

2.  $\sigma_{hh}$  must be linear

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

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

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

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

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

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

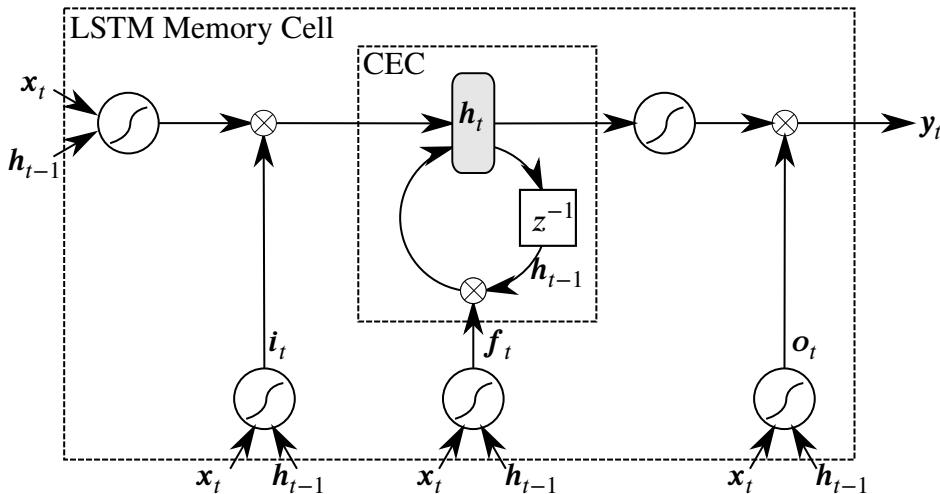


Fig. 2.5 Schematic for a single LSTM memory cell. Notice how the gates  $i_t$ ,  $o_t$ , and  $f_t$  control access to the constant error carousel (CEC).

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

$$2 \quad i_t = \text{sigmoid}(\mathbf{W}_{xi}x_t + \mathbf{W}_{yi}y_{t-1} + \mathbf{b}_i) \quad (2.8)$$

$$3 \quad o_t = \text{sigmoid}(\mathbf{W}_{xo}x_t + \mathbf{W}_{yo}y_{t-1} + \mathbf{b}_o) \quad (2.9)$$

$$4 \quad f_t = \text{sigmoid}(\mathbf{W}_{xf}x_t + \mathbf{W}_{yf}y_{t-1} + \mathbf{b}_f) \quad (2.10)$$

$$5 \quad h_t = f_t \odot h_{t-1} + i_t \odot \tanh(\mathbf{W}_{xh}x_t + \mathbf{W}_{yh}y_{t-1} + \mathbf{b}_h) \quad (2.11)$$

$$6 \quad 7 \quad y_t = o_t \odot \tanh(h_t) \quad (2.12)$$

8 where  $\odot$  denotes elementwise multiplication of vectors.

9 Notice that the gates ( $i_t$ ,  $o_t$ , and  $f_t$ ) controlling flow in and out of the CEC are all time  
10 varying. This can be interpreted as a mechanism enabling LSTM to explicitly learn which error  
11 signals to trap in the CEC and when to release them [72], enabling error signals to potentially  
12 be transported across long time lags.

13 Some authors define LSTM such that  $h_t$  is not used to compute gate activations, referring  
14 to the case where  $h_t$  is connected as “peephole connections” [55]. We will use LSTM to refer  
15 to the system of equations as written above.

## 16 Practicalities for successful applications of LSTM

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

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

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

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

## 2.2 Sequence probability modelling

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

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

1. Modified Markov assumption:

$$\forall t : P(\mathbf{h}_t | \mathbf{h}_{0:t-1}, \mathbf{x}_{1:t}) = P(\mathbf{h}_t | \mathbf{h}_{t-1}, \mathbf{x}_t) \quad (2.13) \quad 20$$

2. Hidden State Stationarity:

$$\forall t_1, t_2 : P(\mathbf{h}_{t_1} = \mathbf{k} | \mathbf{h}_{t_1-1} = \mathbf{i}, \mathbf{x}_{t_1} = \mathbf{j}) = P(\mathbf{h}_{t_2} = \mathbf{k} | \mathbf{h}_{t_2-1} = \mathbf{i}, \mathbf{x}_{t_2} = \mathbf{j}) \quad (2.14) \quad 22$$

3. Output Stationarity:

$$\forall t_1, t_2 : P(\mathbf{y}_{t_1} = \mathbf{j} | \mathbf{h}_{t_1} = \mathbf{i}) = P(\mathbf{y}_{t_2} = \mathbf{j} | \mathbf{h}_{t_2} = \mathbf{i}) \quad (2.15) \quad 24$$

4. Output independence:

$$P(\mathbf{y}_{1:T} | \mathbf{h}_{0:T}, \mathbf{x}_{1:T}) = \prod_{t=1}^T P(\mathbf{y}_t | \mathbf{h}_t, \mathbf{x}_t) \quad (2.16) \quad 26$$

These assumptions imply the sequential factorization:

$$P(\mathbf{y}_{1:T}, \mathbf{h}_{1:T} | \mathbf{h}_0, \mathbf{x}_{1:T}) \quad (2.17)$$

$$= P(\mathbf{y}_{1:T} | \mathbf{h}_{0:T}, \mathbf{x}_{1:T}) P(\mathbf{h}_{1:T} | \mathbf{h}_0, \mathbf{x}_{1:T}) \quad (2.18)$$

$$= \left( \prod_{t=1}^T P(\mathbf{y}_t | \mathbf{h}_t) \right) P(\mathbf{h}_{1:T} | \mathbf{h}_0, \mathbf{x}_{1:T}) \quad \text{eq. (2.16)} \quad (2.19)$$

$$= \left( \prod_{t=1}^T P(\mathbf{y}_t | \mathbf{h}_t) \right) \left( \prod_{t=1}^T P(\mathbf{h}_t | \mathbf{h}_{0:t-1}, \mathbf{x}_{1:t}) \right) \quad (2.20)$$

$$= \left( \prod_{t=1}^T P(\mathbf{y}_t | \mathbf{h}_t) \right) \left( \prod_{t=1}^T P(\mathbf{h}_t | \mathbf{h}_{t-1}, \mathbf{x}_t) \right) \quad \text{eq. (2.13)} \quad (2.21)$$

$$= \prod_{t=1}^T P(\mathbf{y}_t | \mathbf{h}_t, \mathbf{x}_t) P(\mathbf{h}_t | \mathbf{h}_{t-1}, \mathbf{x}_t) \quad (2.22)$$

$$(2.23)$$

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

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

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

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

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

fliang: Draw PGM

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

in

fliang: ref

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

fliang: Define  $V$  to be the vocabulary

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

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

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

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

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

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

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

# 3

## Related Work

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

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

Some common trends throughout related works include

1. Use of domain-specific representations for musical data

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

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

### 3.1 Machine learning on musical data

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

1 structure [6], feature generation for downstream machine learning[33], and style classification  
2 [69].

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

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

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

13 fliang: of what?

14 .

15 Research interests within automatic composition ranges includes generating melody lines  
16 [24, 129, 130], producing harmonizations or accompaniments [41, 70, 132, 4, 2, 21], and  
17 generating full-length novel compositions [46, 119, 44, 117].

18 fliang: emphasize stylistic part

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

## 22 **3.2 Models for automatic composition**

23 Unlike classification, the other tasks (harmonization, completion, automatic composition) re-  
24 quire synthesis of novel music. In a review by Toivainen [131], automatic composition meth-  
25 ods are broadly classified as either symbolic (*i.e.* rule based) or connectionist (*i.e.* neural net-  
26 works).

### 27 **3.2.1 Symbolic rule-based methods**

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

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

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

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

**Experiments in Music Intelligence** (EMI) [28, 26] proposed a system which automatically extracted rules to build an augmented transition network[135]. The system was capable of reproducing music to a particular genre or author, suggesting that the rules extracted by the system can capture a sense of musical style.

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

Genetic algorithms operating on symbolic encodings of music scores have also received significant attention Weinberg et al. [136] and Cope [29]. Emily Howell[27] is an extension of EMI, which uses EMI as a database of compositions to recombine and build novel compositions from[29].

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

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

### 3.2.2 Connectionist methods

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

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

4 fliang: say more here

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

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

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

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

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

### 24 3.2.3 Hybrid methods

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

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

---

### 3.2 Models for automatic composition

**41**

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

fliang: Define this task

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

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

1. Delayed update units to model multiple timescales (described again in Clockwork RNNs [83])
2. Use of Resilient Propagation (RProp) [112] for training ( described again in Liu and Ramakrishnan [89])

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

fliang: Better analyze this

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

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

#### 3.2.4 LSTM music synthesis models

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

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

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

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

### 3.3 Generative modelling of Bach Chorales

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

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

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

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

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

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

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

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

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

Ada Lovelace [16]

# 4

## Automatic composition with deep LSTM

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

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

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

### 4.1 Constructing a corpus of encoded Bach chorales scores

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

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

- 1        2. The large amount of easily recognizable structure: all chorales have exactly four parts
- 2        consisting of a melody in the Soprano part harmonized by the Alto, Tenor, and Bass parts.
- 3        Additionally, each chorale consists of a series of **phrases**: “groupings of consecutive
- 4        notes into a unit that has complete musical sense of its own”[98] which Bach delimited
- 5        using fermatas
- 6        3. The Bach chorales have become a standardized corpus routinely studied by aspiring mu-
- 7        sic theorists[137]

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

#### 10      4.1.1 Preprocessing

11     Motivated by transposition invariance in music (see [section A.1.3](#) on page 112) and prior work  
12     [96, 44, 51, 49], we first perform **key normalization**. The keys of each score were first analyzed  
13     using the Krumhansl Schmuckler key-finding algorithm [84] and then transposed such that the  
14     resulting score is C-major for major scores and A-minor for minor scores.

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

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

22     An example of the distortion introduced through of our preprocessing steps is provided in  
23     [fig. 4.1](#) on the facing page in sheet music notation and in piano roll notation on [fig. 4.2](#) on  
24     page 49.

#### 25      Corpus level analysis of preprocessing effects

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

28     [fig. 4.3](#) plots a histogram of pitch usage counts before and after key normalization. Notice  
29     that the overall range of pitches has increased after key normalization. This can be explained  
30     by noting that Bach’s chorales were to be performed by vocalists and hence were restricted to  
31     use pitches within human voice ranges regardless of key. After transposition, this constraint is

The figure consists of two identical musical staves sets for a four-part chorale. Each set contains four staves labeled from top to bottom: Soprano, Alto, Tenor, and Bass. The music is in common time. The top set of staves represents the original (before preprocessing) and the bottom set represents the processed version. In the original, the Soprano staff has a key signature of one flat (B-flat), while the Bass staff has a key signature of one sharp (F-sharp). After preprocessing, both the Soprano and Bass staves have a key signature of one sharp (F-sharp), and the Alto and Tenor staves remain in one flat (B-flat). The music consists of four measures of a chorale tune.

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

- <sup>1</sup> no longer be satisfied and we see the appearance of unrealistically low notes (*e.g.* A1) outside
- <sup>2</sup> the range of even the lowest voice types.

## 4.1 Constructing a corpus of encoded Bach chorales scores

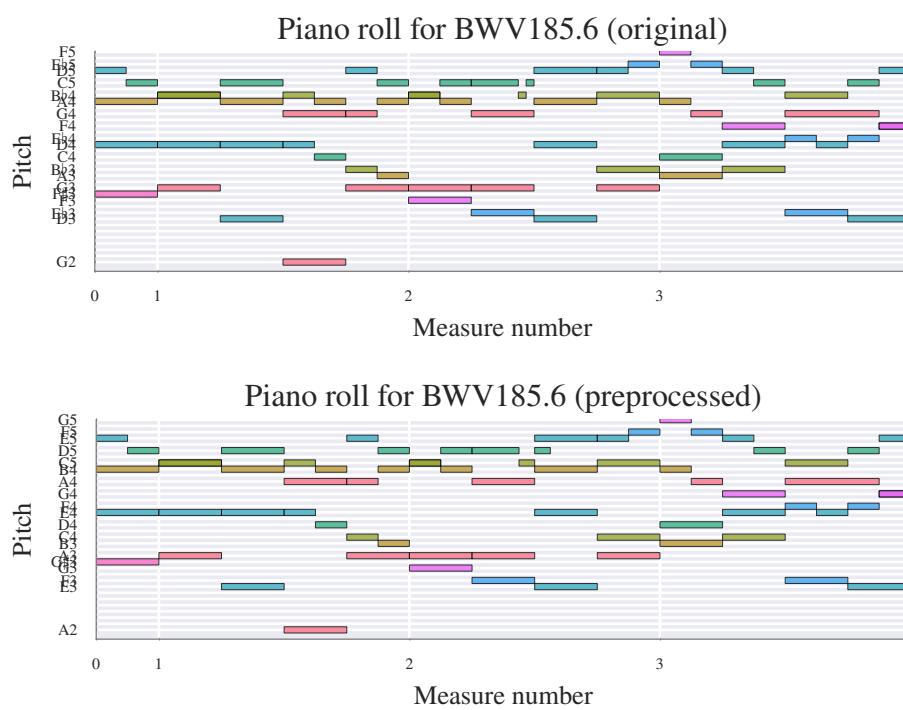
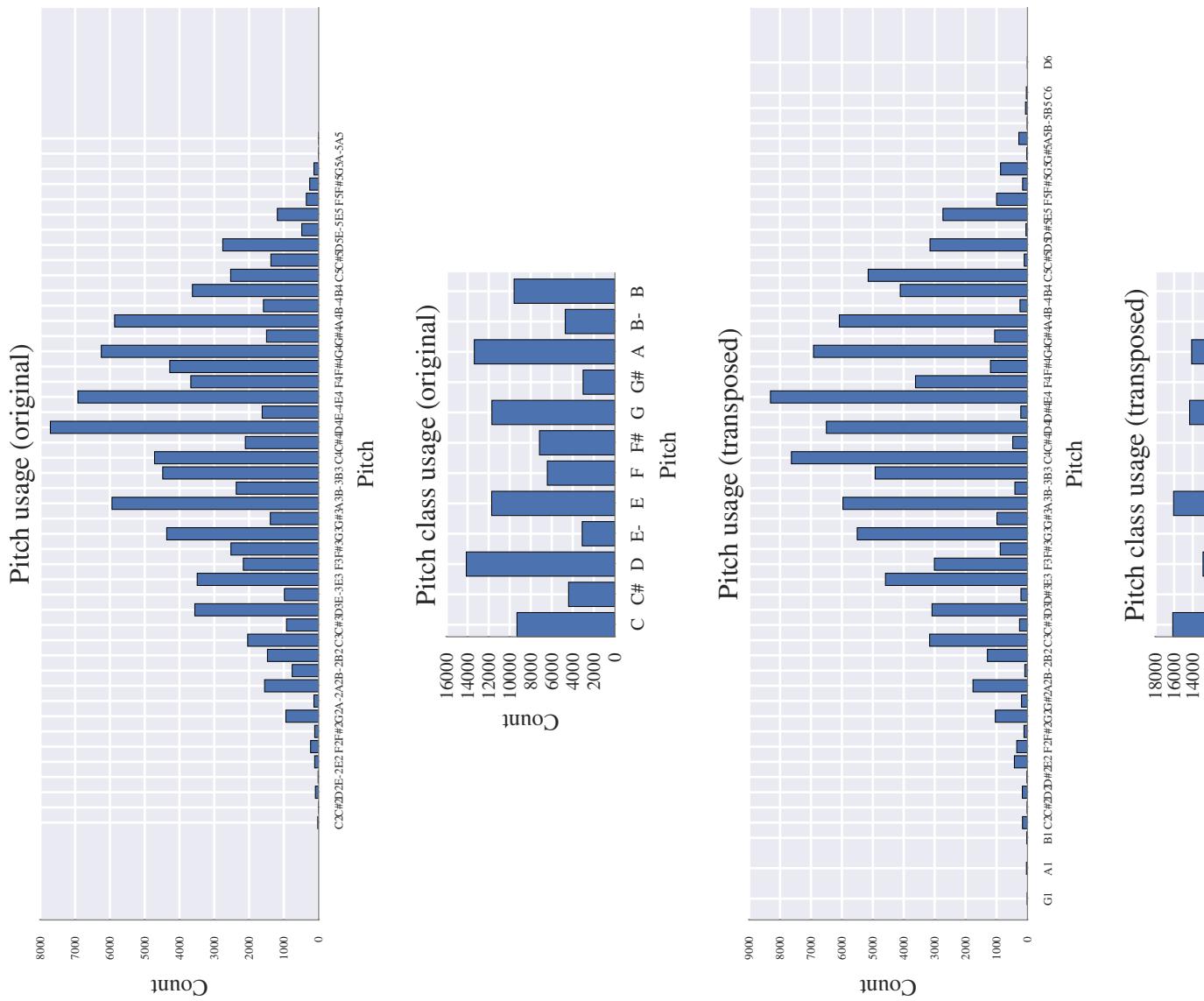


Fig. 4.2 Piano roll representation of the same 4 bars from [fig. 4.1](#) before and after preprocessing. Again, note the transposition to C-major and time-quantization occurring in the Soprano part.



## 4.1 Constructing a corpus of encoded Bach chorales scores

51

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.

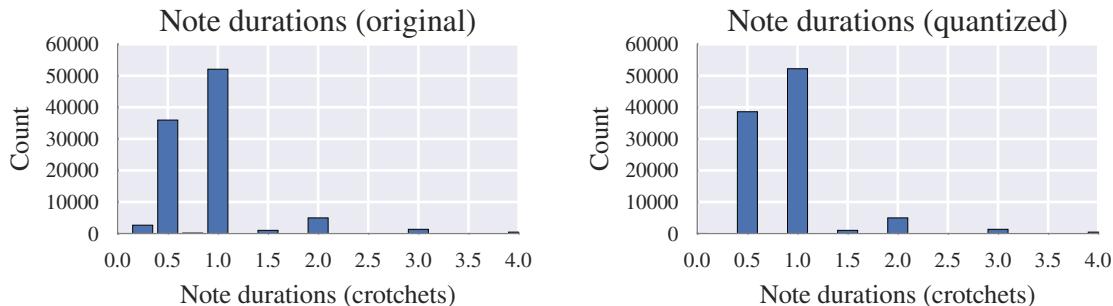


Fig. 4.5 Distribution of note durations over Bach chorales corpus. Quantization has minimal impact because of the high resolution (semiquavers) used.

In fig. 4.4, 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 fig. 4.5, 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 [96, 51, 123]) or larger harmonic units (*e.g.* chords [44, 15], “harmonic context” [2]). This tradeoff is similar to one faced in RNN language modelling where either individual characters or entire words can be used.

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

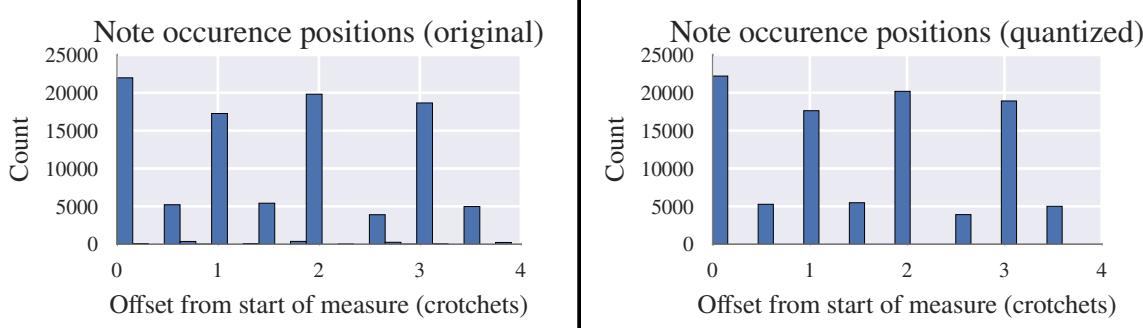


Fig. 4.6 Meter is minimally affected by quantization due to the high resolution used for time quantization.

1           fiang: reference

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

6         Similar to [130], we represent polyphonic scores using a localist frame-based representa-  
7         tion where time is discretized into constant timestep **frames**. Frame based processing forces the  
8         network to learn the relative duration of notes, a counting and timing task which [57] demon-  
9         strated LSTM is capable of. Consecutive frames are separated by a unique delimiter (“|||” in

10          fiang: Figure of score encoded in text

11           ).

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

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

21           fiang: reference

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

## 4.1 Constructing a corpus of encoded Bach chorales scores

Table 4.1 Statistics on the preprocessed datasets used throughout our study

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

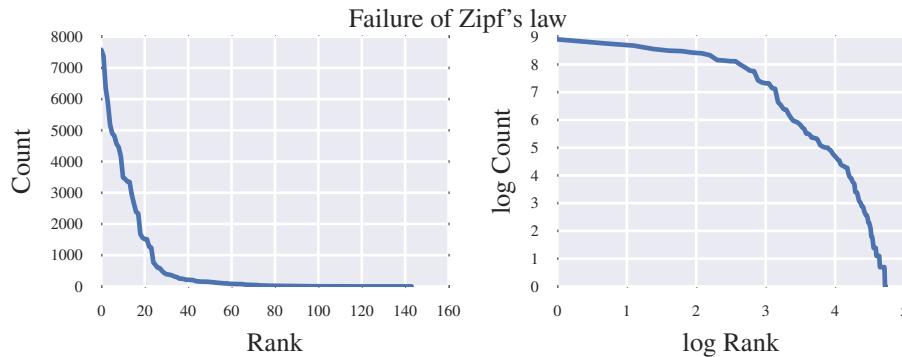


Fig. 4.7 Left: Token frequencies sorted by rank. Right: log-log plot where a power law distribution as predicted by Zipf's law would appear linear.

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 respectively. This causes the model to learn to initialize itself when given the start symbol and allows us to determine when a composition generated by the model has concluded. The vocabulary and corpus size after encoding is detailed in [table 4.1](#). The rank-size distribution of the note-level corpus tokens is shown in [fig. 4.7](#) 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 where multiple articulations at the same pitch are indistinguishable from a single note with the same duration, an issue present in many prior works [[44](#), [43](#), [89](#), [17](#)].

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

- <sup>1</sup> intentional and is motivated by numerous reports [11][12] suggesting that that a key ingredient  
<sup>2</sup> in deep learning's success is its ability to learn good features from raw data.

## <sup>3</sup> 4.2 Design and validation of a generative model for music

<sup>4</sup> In this section, we describe the design and validation process leading to our generative model.  
<sup>5</sup> Unlike many prior models for music data, we intentionally avoid injection of domain-specific  
<sup>6</sup> knowledge into our model architectures such as distinguishing between chords versus notes  
<sup>7</sup> [70][96] [44] and explicitly modelling of meter [43] or motifs [47]. Through this fundamental  
<sup>8</sup> connectionist approach, we aim to minimize biases introduced by prior assumptions and force  
<sup>9</sup> the model itself to learn musical structure from data.

### <sup>10</sup> 4.2.1 Training and evaluation criteria

<sup>11</sup> Following [96], we will train the model to predict a distribution distribution over all possible to-  
<sup>12</sup> kens next  $\mathbf{x}_{t+1}$  given the current token  $\mathbf{x}_t$  and the previous hidden state  $\mathbf{h}_{t-1}$ . This is equivalent  
<sup>13</sup> to setting the target sequence to be the input sequence delayed by one timestep:  $\mathbf{y}_{1:T-1} = \mathbf{x}_{2:T}$   
<sup>14</sup> and  $\mathbf{y}_T = \text{STOP}$ .

<sup>15</sup> fliang: Diagram for sequential prediction

<sup>16</sup> .  
<sup>17</sup> fliang: Note similarity with language modeling

<sup>18</sup> .  
<sup>19</sup> For training criteria, we use the cross-entropy between the predicted distributions  $P(\mathbf{y}_t | \mathbf{h}_t, \mathbf{x}_t)$   
<sup>20</sup> and the actual target distribution  $\delta_{\mathbf{y}_t}$ .

<sup>21</sup> Note that our training criteria as written in

<sup>22</sup> fliang: reference

<sup>23</sup> uses the actual next token  $\mathbf{x}_{t+1}$  as the recurrent input, even if the most likely prediction  
<sup>24</sup> argmax  $P(\mathbf{x}_{t+1} | \mathbf{h}_t, \mathbf{x}_t)$  differs. This is referred to as **teacher forcing**[139] and is motivated  
<sup>25</sup> by the observation that model predictions may not yet be correct during the early iterations  
<sup>26</sup> of training. However, at inference the token generated from  $P(\mathbf{x}_{t+1} | \mathbf{h}_t, \mathbf{x}_t)$  is reused as the  
<sup>27</sup> previous input, creating a discrepancy between training and inference. Scheduled sampling  
<sup>28</sup> [10] is a recently proposed alternative training method for resolving this discrepancy and may  
<sup>29</sup> help the model better learn to predict using generated symbols rather than relying on ground  
<sup>30</sup> truth to be always provided as input.

### 4.2.2 Establishing a baseline with $N$ -gram language models

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

Both models were developed for applications modelling language data, whose distribution over words which may differ from our encoded music data (see [fig. 4.7](#) on page [53](#)). Furthermore, both are based upon  $N$ -gram models which are constrained to only account for short-term dependencies. Hence, we expect RNNs to outperform the results shown in [table 4.2](#) on the next page.

### 4.2.3 Description of RNN model hyperparameters

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

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

fliang: Does this need to be diagrammed?

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

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

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

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

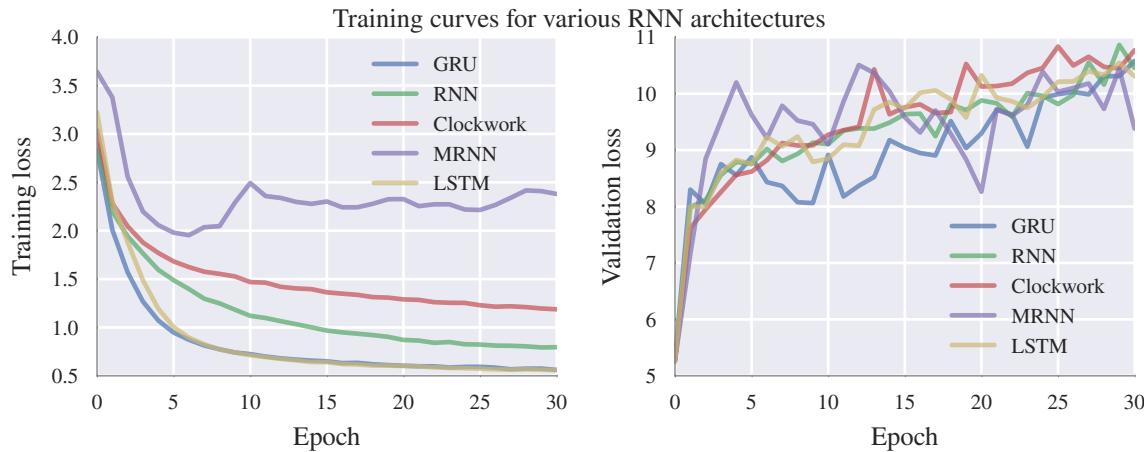


Fig. 4.8 LSTM and GRUs yield the lowest training loss. Validation loss traces show all architectures exhibit signs of significant overfitting

#### 4.2.4 Comparison of memory cells on music data

To rapidly compare a large number of memory cell implementations, we leveraged [theanets](#)<sup>1</sup>: a Python software library for high-level specification of neural network models. [Figure 4.8](#) shows the results of exploring a range of RNN memory cell implementation and holding `num_layers=1`, `rnn_size=130`, `wordvec=64`, and `seq_length=50` constant. Unlike later models, none of these models utilized dropout or batch normalization. We used a batch size of 50 and configured the clockwork RNN [20] with 5 equal-sized hidden state blocks with update periods (1, 2, 4, 8, 16).

[Figure 4.8](#) shows that while all models achieved similar validation losses, LSTM and GRUs trained much faster and achieved lower training loss. Since Zaremba [143] find similar empirical performance between LSTM and GRUs and Nayebi and Vitelli [99] observe LSTM outperforming GRUs in music applications, we choose to use LSTM as the memory cell for all following experiments.

The increasing validation loss over time in [fig. 4.8](#) is a red flag suggesting that overfitting is occurring. This observation motivates the exploration of dropout regularization in [section 4.2.5](#).

#### 4.2.5 Optimizing the LSTM architecture

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

<sup>1</sup><https://github.com/lmjohns3/theanets>

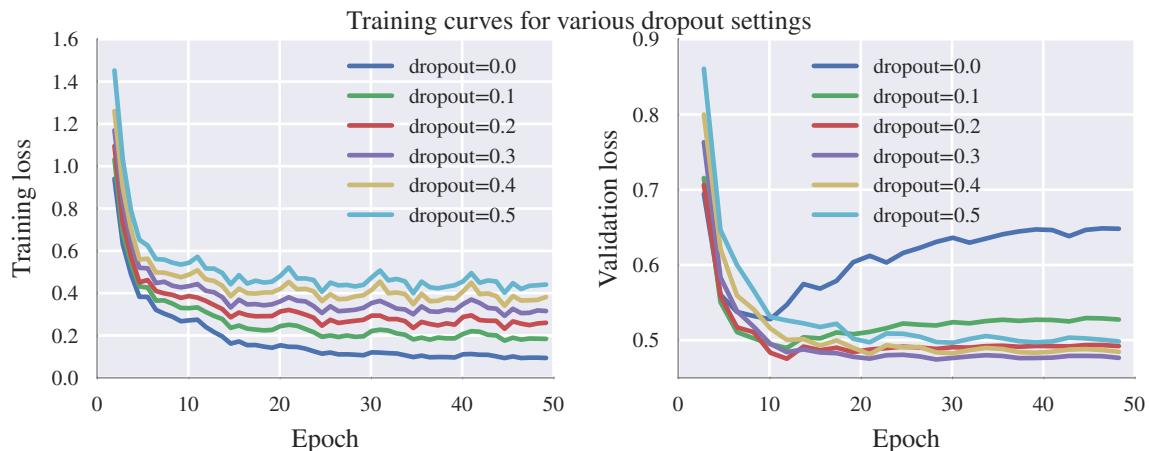


Fig. 4.9 Dropout acts as a regularizer, resulting in larger training loss but better generalization as evidenced by lower validation loss. A setting of dropout=0.3 achieves best results for our model.

### <sup>1</sup> Dropout regularization improves validation loss

<sup>2</sup> fliang: Is it still worth it to do some batch normalization experiments?

<sup>3</sup> The increasing validation errors in fig. 4.8 on page 57 prompted investigation of regularization techniques. In addition to adding batch normalization, a technique known to reduce <sup>4</sup> overfitting and accelerate training [77], we also investigated the effects of different levels of <sup>5</sup> dropout by varying the dropout parameter.

<sup>6</sup> The experimental results are shown in fig. 4.9. As expected, dropout acts as a regularizer <sup>7</sup> and reduces validation loss from 0.65 down to 0.477 (when dropout=0.3). Training loss has <sup>8</sup> slightly increased, which is also unexpected as application of dropout during training intro- <sup>9</sup> duces additional noise into the model.

### <sup>11</sup> Overall best model

<sup>12</sup> We perform a grid search through the following parameter grid:

- <sup>13</sup> • num\_layers ∈ {1, 2, 3, 4}
- <sup>14</sup> • rnn\_size ∈ {128, 256, 384, 512}
- <sup>15</sup> • wordvec ∈ {16, 32, 64}
- <sup>16</sup> • seq\_length ∈ {64, 128, 256}
- <sup>17</sup> • dropout {0.0, 0.1, 0.2, 0.3, 0.4, 0.5}

## 4.2 Design and validation of a generative model for music

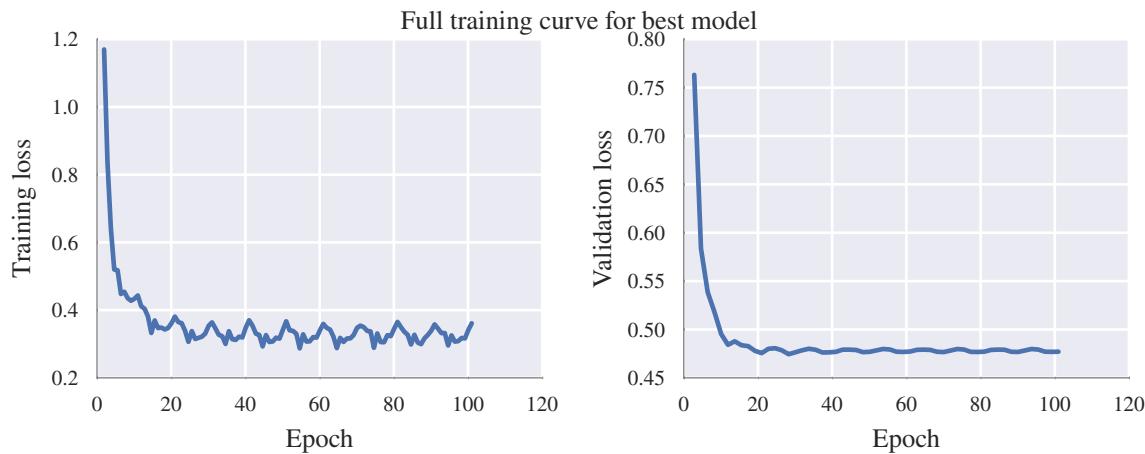


Fig. 4.10 Training curves for the overall best model. The periodic spikes correspond to resetting of the LSTM state at the end of a training epoch.

Table 4.3 Timing results comparing CPU and GPU training of the overall best model (section 4.2.5 on page 58)

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

A full listing of results is provided in fig. B.1 on page 117.

The optimal hyperparameter settings within our grid was found to be `num_layers = 3`, `rnn_size =`, `wordvec = 32`, `seq_length = 128` `dropout = 0.3`. Such a model achieves 0.324 and 0.477 cross entropy losses on training and validation corpuses respectively. Figure 4.10 plots the training curve of this model and shows that training converges after only 30 iterations.

To confirm local optimality, we perform perturbations about our final hyperparameter settings in figs. B.2 to B.6.

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

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

- Larger `rnn_size` leads to higher capacity and lower training loss

- Presents as overfitting on validation, where the lowest capacity model `rnn_size` appears to be improving in generalization while others are flat/increasing

- Training curves about the same wrt `num_layers`, validation curves have interesting story
  - Depth matters: small 64 and 128 hidden unit RNNs saw improvements up to 0.09
  - Unsurprising, Pascanu et al. [104] already observed deep RNNs dominated polyphonic modelling.
  - Expressivity gained from depth furthers overfitting: 256 hidden unit RNN has some of the best validation performance at depth 1 but is the worst generalizing model for depths 2 and 3 even though training loss is low
- `rnn_size=128` found to be best generalizing, optimized at `num_layers=2`: will continue with these settings

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

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

## 4.3 Results

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

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

Our best model achieves cross-entropy losses of 0.323 on training data and 0.477 on held-out test data (fig. B.1 on page 117), corresponding to a training perplexity of 1.251 bits and a test perplexity of 1.391. This is more than 0.6 bits lower than any test perplexity obtained by the  $N$ -gram models compared in table 4.2 on page 56. This result is expected and shows that the greater modelling capacity provided by RNNs is useful for encoded music score data.

fliang: Compare [17]

1

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

2

3

4

5

6

7

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

8

## 4.4 Other applications

9

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

10

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

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

Hoffmann [73]

# 5

## Analysis of musical concepts learned by the model

### 5.1 Investigation of neuron activation responses to applied stimulus

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

We use as stimulus the music score shown in [fig. 5.1](#), which has been preprocessed according to

fliang: cite preprocessing

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

In [fig. 5.2](#) we visualize the network activations as the stimulus is sequentially applied. Note that as a consequence of the variable-length encoding format described in

fliang: ref

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

### 5.1.1 Pooling over frames

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

fliang: cite

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

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

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

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

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

The max-pooled frame-level activations are shown in [fig. 5.3](#). As a result of pooling, the horizontal axis can be aligned and compared against the stimulus [fig. 5.1](#). Notice the appearance of vertical bands corresponding to when a chord/rest is held for multiple frames. In particular, the vector embedding corresponding to rests (*e.g.* near frames 30 and 90 in [fig. 5.3](#) top) are sparse, showing up as white smears not only in the embedding layer but on all LSTM memory cells.

### 5.1.2 Probabilistic piano roll: likely variations of the stimulus

The bottom panel in [fig. 5.3](#) shows the model’s predictions for tokens in the next frames, where the tokens are arranged according to (arbitrary) index within the vocabulary. As the tokens correspond to pitches, they can be sorted according to pitch to reconstruct a **probabilistic piano roll**[\[43\]](#) consisting of the model’s sequence of next-frame predictions as it processes the input.

## 5.1 Investigation of neuron activation responses to applied stimulus

65

fliang: Align these

Notice that the probabilistic piano roll in [fig. 5.4](#) closely resembles the stimulus. This is unsurprising because the recurrent inputs are taken from the stimulus rather than sampled from the model’s predictions (a.k.a. [139]), so a model which predicts to only continue holding its input would produce a probabilistic piano roll identical to the stimulus delayed by one frame.

Two interesting rows of [fig. 5.4](#) are the rows corresponding to frame delimiters (fourth from top, “|||”) and fermatas (third from top “(.)”). Notice that the predictions for chord delimiters are particularly strong during rests. This is because rests are encoded as empty frames, so the large probability values indicate that the model has learned to prolong periods of rests. At the end of rest periods, the model tends to assign probability across a wide range of notes, consistent with the intuition that the possible notes occurring directly after a rest is less constrained than

fliang: cite the intuition?

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

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

## 5.1.3 Neurons specific to musical concepts

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

fliang: Cite deconvolution

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

To investigate this further, we look at the activations over time of individual neurons within the LSTM memory cells. Our results confirm our hypothesis: we discover certain neurons whose activities are correlated to specific motifs, chord progression, and phrase structures. The activity profiles of these neurons are shown in [fig. 5.5](#).

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

The first three neurons  $((1, 64), (1, 138), (1, 207))$  shown in the 2nd to 4th panel from top of [fig. 5.5](#) 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

<sup>1</sup> chord inversions which they are most sensitive to. In contrast, (1, 207) only exhibits activity  
<sup>2</sup> for the *V* chord associated with the imperfect cadences near frames 150 and 180.

<sup>3</sup> The next two neurons in [fig. 5.5](#), (1, 87) and (1, 151), act as motif detectors. Activity in  
<sup>4</sup> (1, 151) peaks when a *vi – vii – vi* progression is present in the stimulus. (1, 87) exhibits large  
<sup>5</sup> spikes on *I – –V – –I*,

<sup>6</sup> (2, 37) exhibits less specificity, but has large spikes right before the *IV* chord in *I – –IV*  
<sup>7</sup> chord progressions.

<sup>8</sup> (2, 82) peaks at the top of an ascending harmonic progressions, right before a descending  
<sup>9</sup> major scale is to follow.

<sup>10</sup> (2, 243) is specific to *v – –vi* progressions, with large spikes occurring at the *v – –vi* pro-  
<sup>11</sup> gressions near frames 55, 120, 130, and a lower intensity spike at 170. Some activity is also  
<sup>12</sup> observed for the *V – –vi* around frame 230 despite the first chord being a major mode *V* rather  
<sup>13</sup> than minor *v*.

## 5.1 Investigation of neuron activation responses to applied stimulus

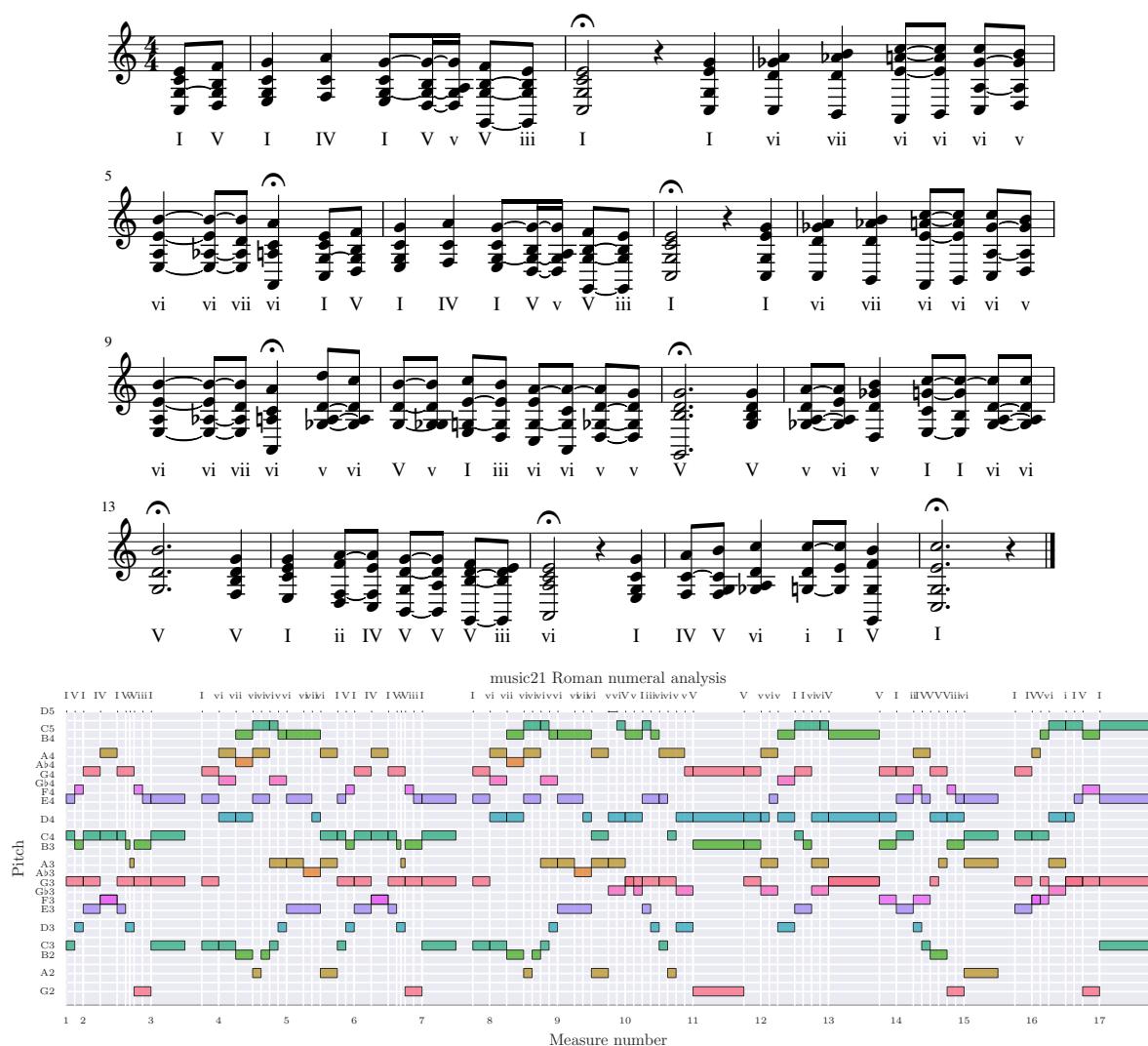


Fig. 5.1 *Top*: The preprocessed score (BWV 133.6) used as input stimulus with Roman numeral analysis annotations obtained from music21; *Bottom*: The same stimulus represented on a piano roll

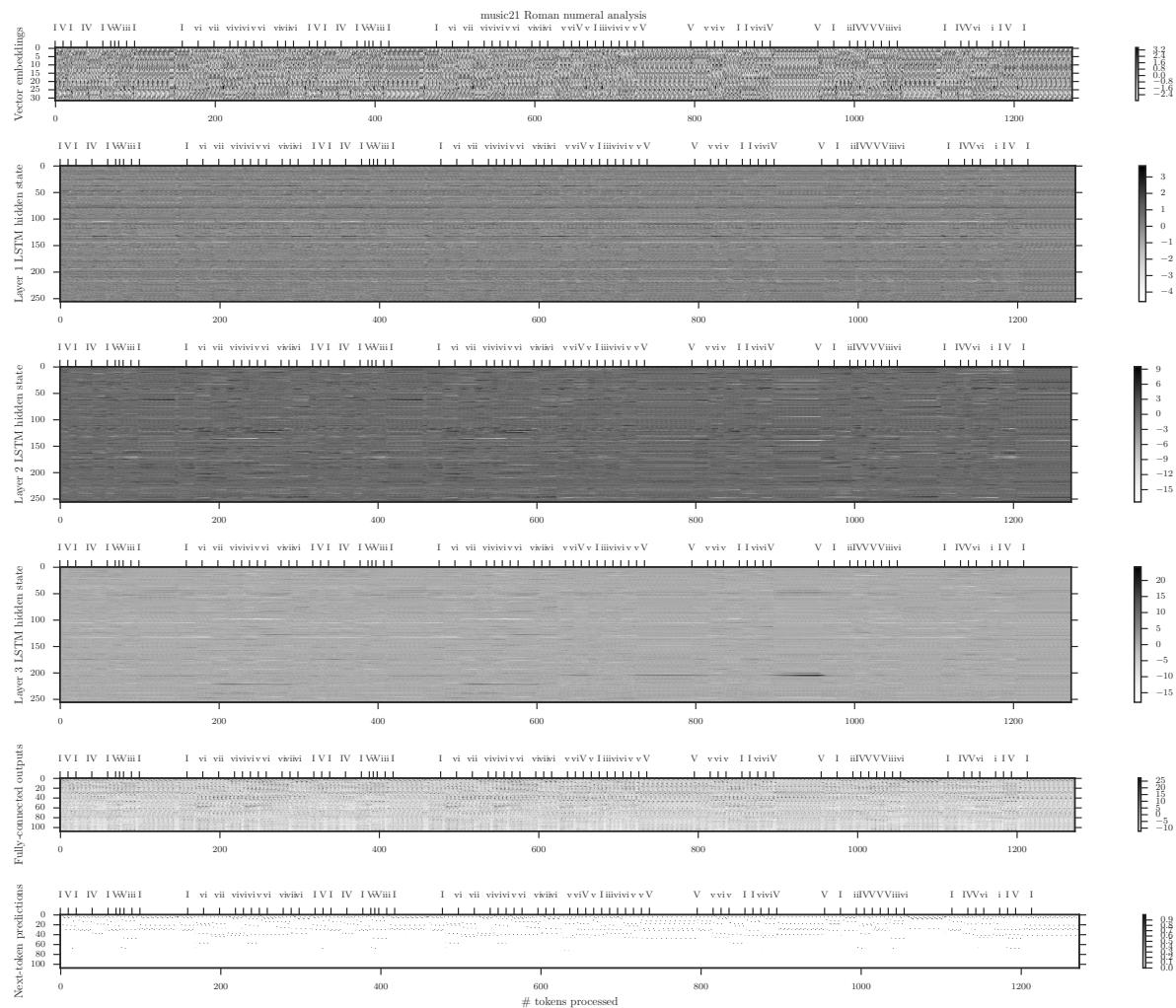


Fig. 5.2 Neuron activations over time as the encoded stimulus is processed token-by-token

## 5.1 Investigation of neuron activation responses to applied stimulus

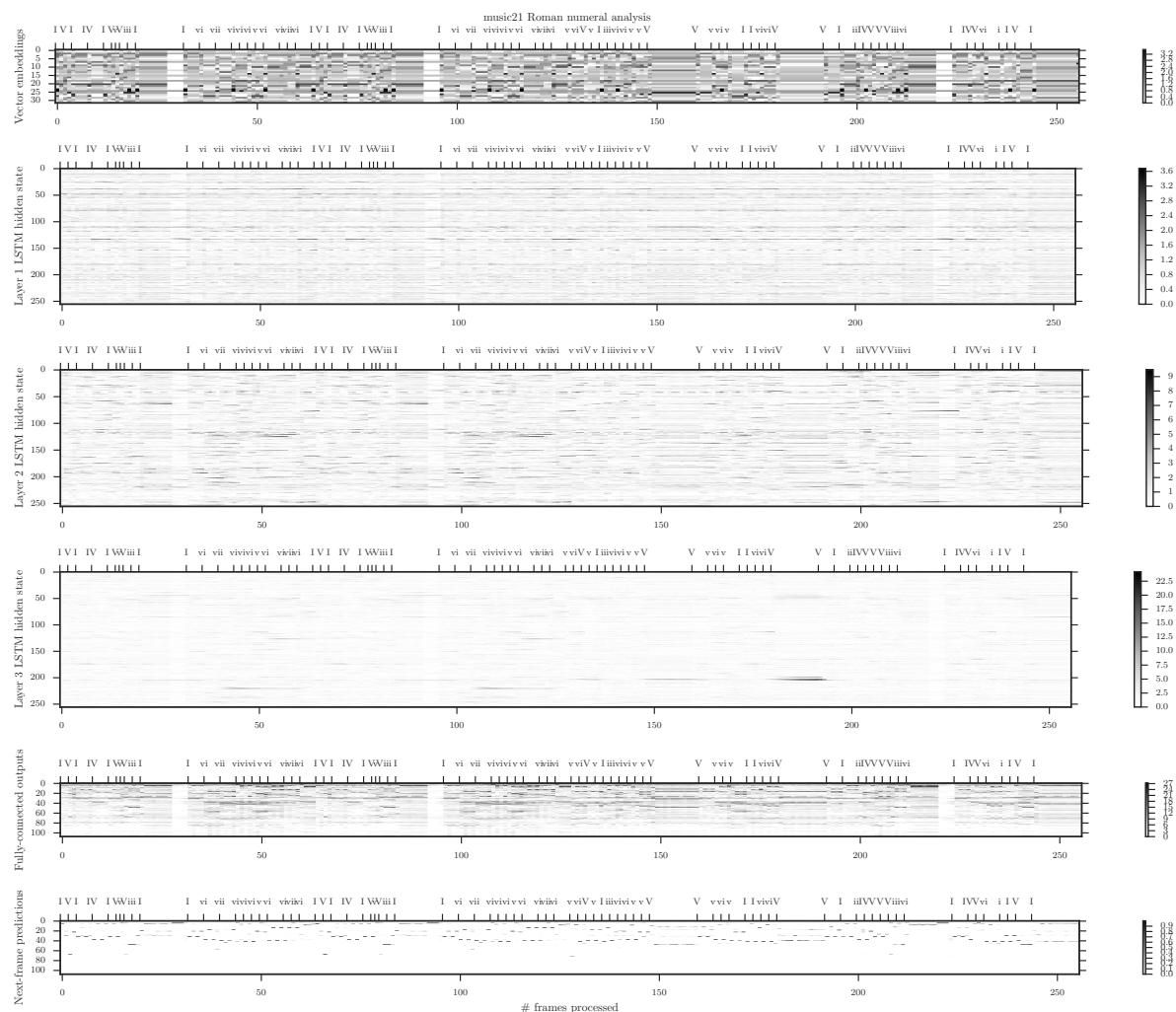


Fig. 5.3 Neuron activations over time pooled over frames

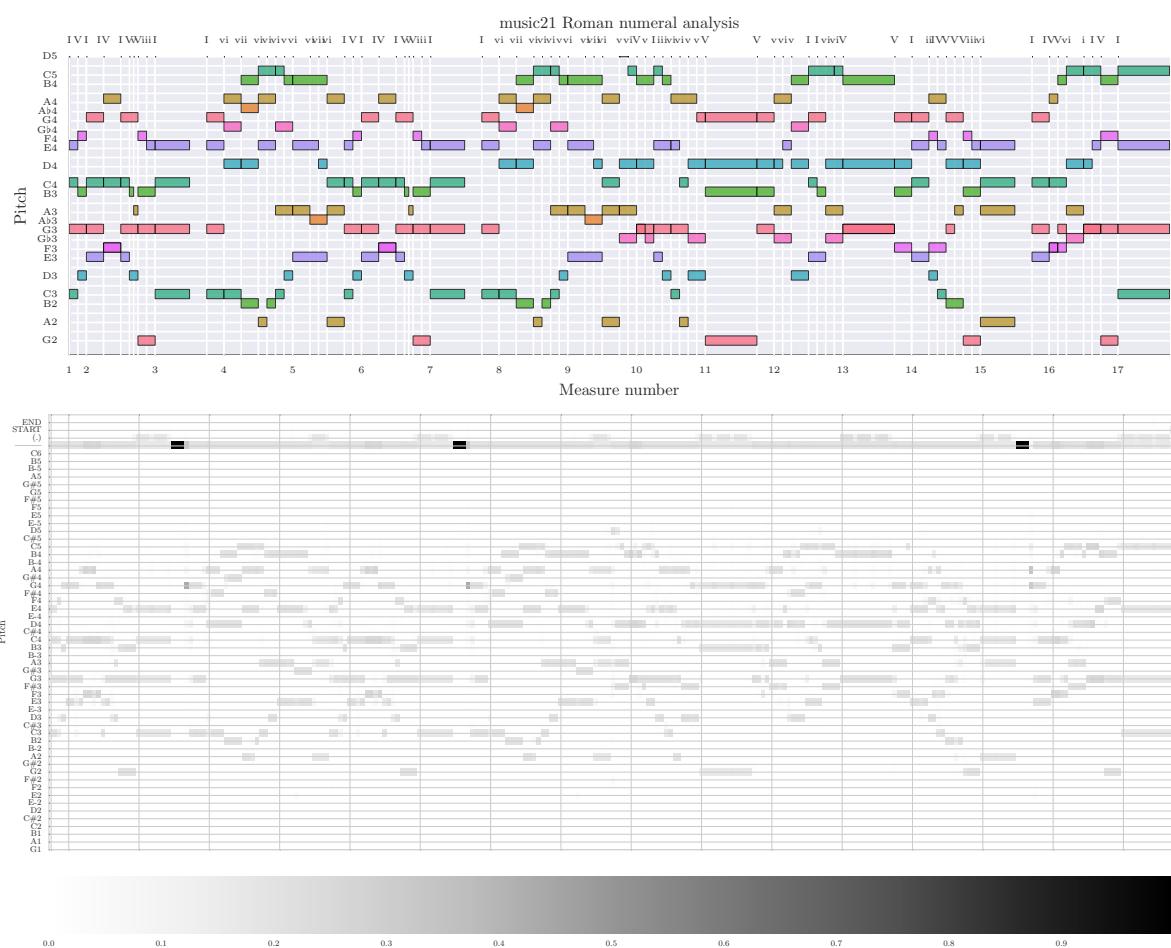


Fig. 5.4 Top: piano roll of stimulus (included for reference); Bottom: probabilistic piano roll

## 5.1 Investigation of neuron activation responses to applied stimulus

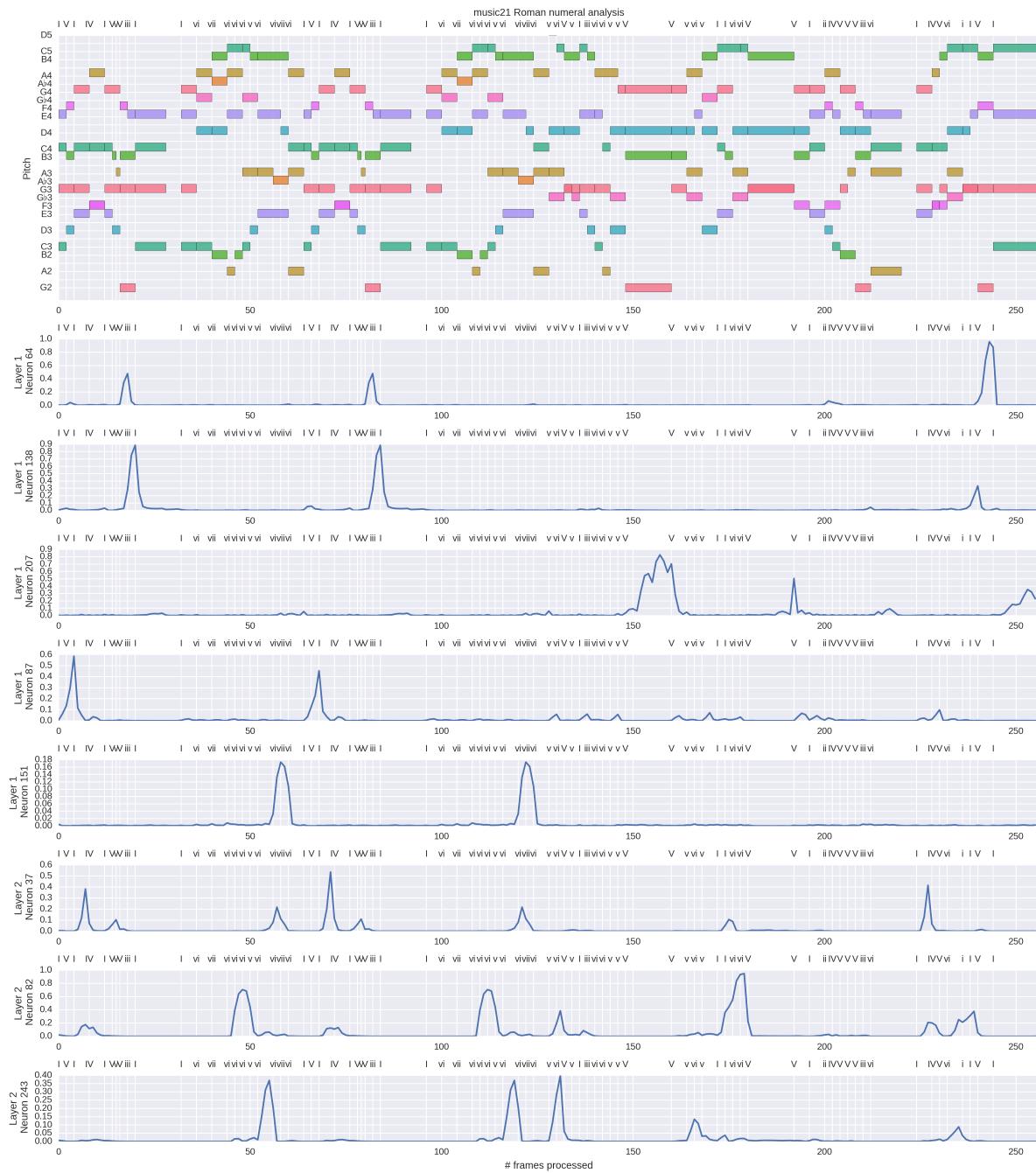


Fig. 5.5 Activation profiles of neurons within our model which have learned high-level musical concepts

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

# 6

## Chorale harmonization

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

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

### 6.1 Background

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

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

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

For a music student studying chorale harmonization, a common pedagogical exercise [36][107] is

a sequence of tasks increasing in difficulty:

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

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

## 6.2 Harmonizing

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

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

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

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

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

This solution is approximate because while the factorization 1

$$P(w_{1:L}) = \prod_{t=1}^L P(w_t | w_{1:t-1}) \quad (6.3) \quad 2$$

is true and justifies our model, the factorization 3

$$P(w_{1:L} | w_\alpha) = \prod_{t=1}^L \hat{P}(w_t | w_{1:t-1}) \quad (6.4) \quad 4$$

does not hold. Some primary criticisms include 5

- 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) 6
- 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 8
- Assumption that prior selections  $w_{1:t-1}$  optimize  $P(w_t | w_{1:t-1})$ : the model  $\hat{P}$  is trained on data which assumes all prior inputs have been ground truth. It has been shown 10

fliang: mikolov 13

that such an assumption can lead to very sensitive hidden state dynamics which are not robust to errors (i.e. when  $w_{1:t-1}$  contain errors). 14

Beam search is one way to mitigate the effects of greedy selection: our current method is equivalent to a beam search with width one. Maintaining  $N$ -best hypotheses using a lattice-based framework such as in Liu et al. [90] would allow the model to partially recover from mistakes made during greedy action selection. 16

Furthermore, another potential issue with the proposed model is a discrepancy between the inputs provided during training (which are taken from actual data sequences regardless of model predictions) and sampling (where the inputs are generated by the model at the previous timestep). To resolve this discrepancy, Bengio et al. [10] proposed **scheduled sampling** as a curriculum learning strategy gradually transitioning sequence models to rely on their prior predictions as inputs and thereby learn state dynamics more resilient to erroneous inputs. 20

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. 26

1

2

3

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

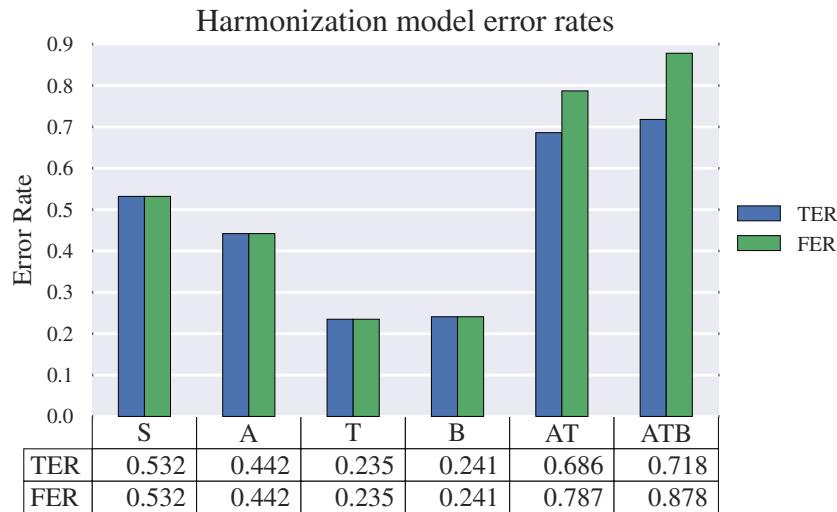


Fig. 6.1 Error rates for harmonization tasks

## <sup>1</sup> 6.3 Datasets

<sup>2</sup> We create datasets where one or more parts are masked:

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

<sup>6</sup> Of particular interest is the AT dataset. Bach oftentimes only wrote the Soprano and Bass  
<sup>7</sup> parts of a piece, leaving the middle parts to be filled in by students. Our networks performance  
<sup>8</sup> on this task can be used as a benchmark for an easier task. Another interesting configuration  
<sup>9</sup> is ATB, which corresponds to harmonizing a given melody (*i.e.* Soprano line) and can be  
<sup>10</sup> compared against prior work such as Allan and Williams [2].

## <sup>11</sup> 6.4 Results

<sup>12</sup> We are unable to compare against Allan and Williams [2] because their model consists of mul-  
<sup>13</sup> tiple domain specific subtasks (*e.g.* harmony and chord skeletons, ornamentation) and because  
<sup>14</sup> they only consider the ATB case.

## 6.4 Results

6

S.

A.

T.

B.

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

**6.4.1 Harmonizing popular tunes with BachBot**

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

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

1  
2  
3  
4  
5

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

# 7

1

## Large-scale subjective evaluation

2

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

3

4

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

5

6

7

8

9

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

10

11

12

13

14

[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.

15

16



## Challenge description

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

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

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

## <sup>1</sup> 7.1 Evaluation framework design

### <sup>2</sup> 7.1.1 Software architecture

<sup>3</sup> The frontend utilizes React and Redux, allowing us to collect fine-grained user action data.  
<sup>4</sup> Azure App Service is used to host an Express web-service which randomizes experimental  
<sup>5</sup> questions and collects responses. The data is stored to Azure Data Storage and processed in  
<sup>6</sup> batch MapReduce using Azure HDInsight.

### <sup>7</sup> 7.1.2 User interface

<sup>8</sup> The landing page for <http://bachbot.com/> is shown in fig. 7.1.

<sup>9</sup> Clicking “Test Yourself” redirects the participant to a user information form (fig. 7.2) where  
<sup>10</sup> users self-report their age group prior music experience into the categories shown.

## Some background info about you

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

**Self-rating of music experience**

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

**Submit**

**Clear Values**

Fig. 7.2 User information form presented after clicking “Test Yourself”

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

### 7.1.3 Question generation

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

fliang: ref

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

### 7.1.4 Promoting the study

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

## The BachBot Challenge

The screenshot shows a user interface for a music recognition challenge. At the top, a header reads "Select the music most similar to Bach". Below this is a "Select" button and a set of playback controls (play/pause, previous, next, volume). The main area contains another "Select" button and playback controls. In the bottom right corner of this area is a blue "Submit" button. Below the main area is a horizontal progress bar with a yellow segment indicating "40%" completion. At the bottom center, the text "Question 2 out of 5" is displayed.

Fig. 7.3 Question response interface used for all questions

DATE 1 ing. We found that 52.1% of participants were referred from social media, 4.8% through other  
2 websites and blogs, 2% through search engine results, and the remaining 41.2% had directly  
3 accessed [bachbot.com](http://bachbot.com),

## 4 7.2 Results

### 5 7.2.1 Participant backgrounds and demographics

6 fliang: Update with most recent results before submitting

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

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

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

As evidenced by fig. 7.4 on the following page, our participant is diverse and includes participants from six different continents. fig. 7.5 on page 85 shows that while the majority of our participants are between 18 – 45 and have played an instrument, more than 23.7% have either formally studied or taught music theory.

## 7.2.2 BachBot's performance results

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

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

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

fliang: VERIFY LAST

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

fliang: VERIFY LAST

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

fliang: Collins only uses expert evaluators

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

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

fliang: reference

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

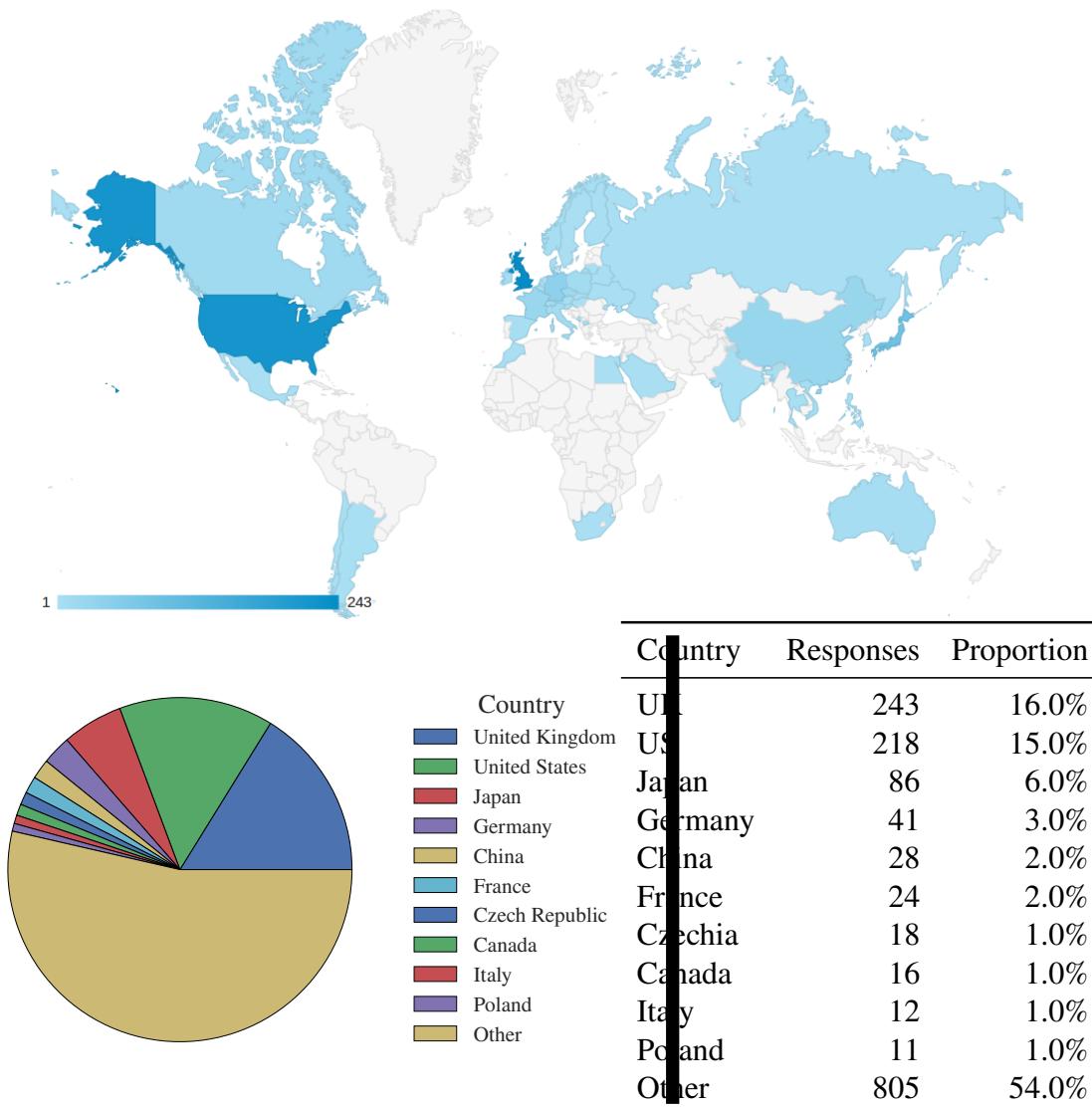


Fig. 7.4 Geographic distribution of participants

## 7.2 Results

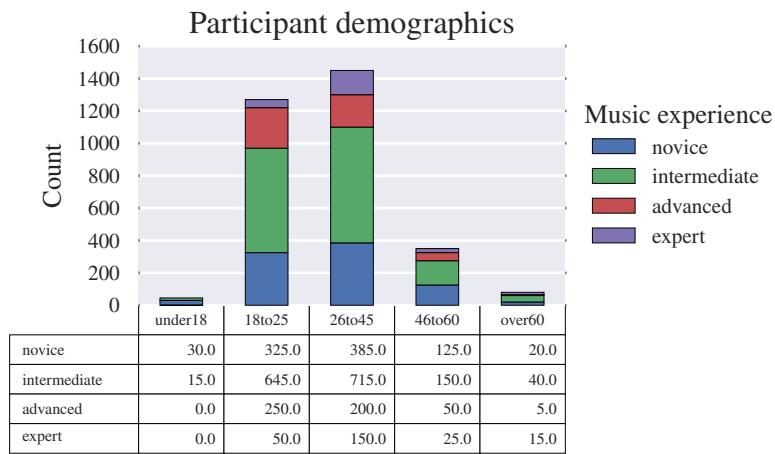


Fig. 7.5 Demographics of participants

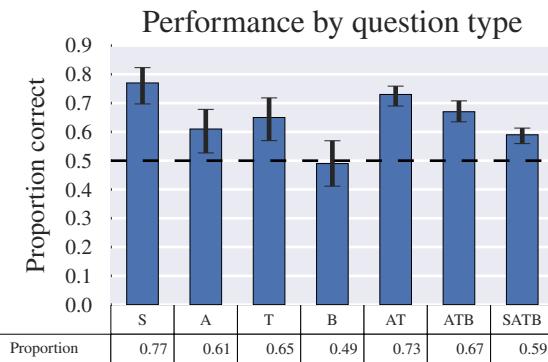


Fig. 7.6 responses-Mask

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

fliang: cite

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

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

fliang: cite

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

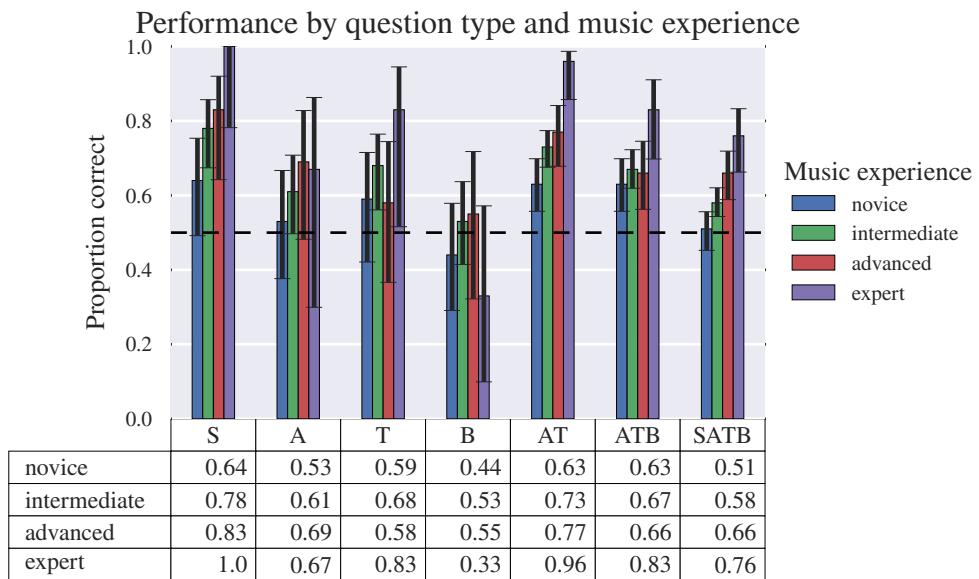


Fig. 7.7 Proportion of correct responses for each question type and music experience level.

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

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

fliang: Comment more about these figures in captions

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

fig. 7.9 shows the proportion correct for each question. Encouragingly, it shows that 41.7%

fliang: VERIFY LAST

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

fliang: Have Mark analyze bad examples in fig. 7.9

### 7.3 User feedback

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

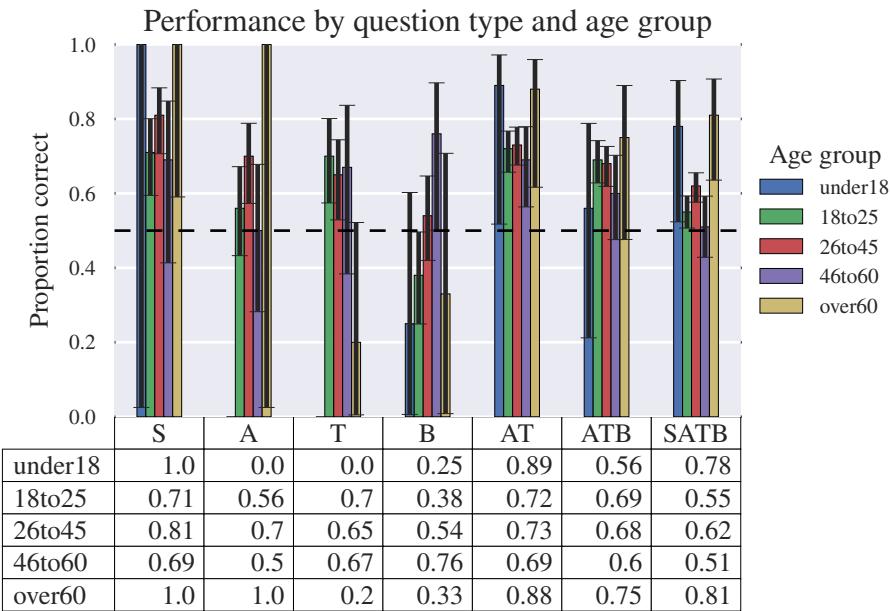


Fig. 7.8 Proportion of correct responses for each question type and age group.

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

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

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

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

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

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

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

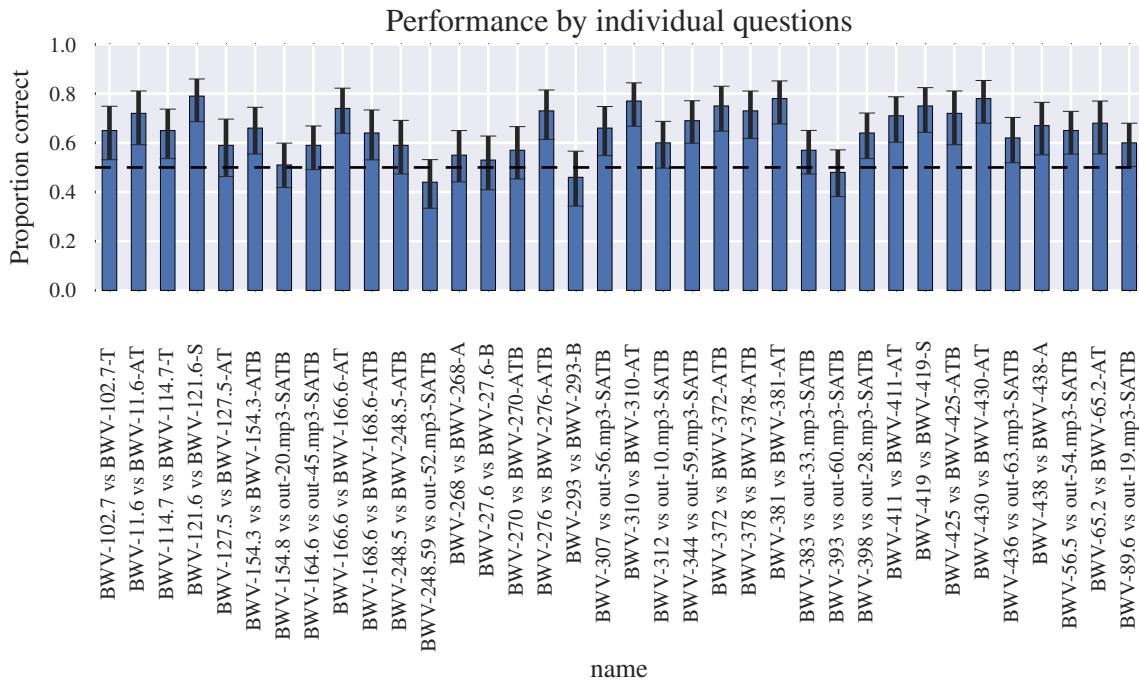


Fig. 7.9 Proportion of correct responses broken down by individual questions.

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

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

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

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

The whole thing is really quite impressive overall though...

## 7.4 Competitive analysis of large-scale evaluation methodologies

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

fliang: Compare costs and quality with MTurk

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

fliang: UPDATE THIS NUMBER

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

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

[40]

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

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

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

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

Bharucha and Todd [14]

# 8

## Discussion, Conclusions, and Future Work

### 8.1 Discussion

#### 8.1.1 Contributions

- An encoding scheme for representing music with arbitrary degrees of polyphony as ordered sequences of tokens 5  
6
- 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 7  
8  
9
- Optimized the performance of the proposed model and quantitatively evaluated its performance on both composition and harmonization tasks 10  
11
- Performed the largest (as of Thursday 11<sup>th</sup> August, 2016) published musical Turing test of an automatic composition system 12  
13
- Investigated the internal representations learned by the model, identifying neurons specific to music-theoretic concepts. 14  
15

## <sup>1</sup> 8.2 Conclusions

<sup>2</sup> Recall that our journey was prompted by the questions: can the current state-of-the-art in  
<sup>3</sup> deep learning be used to build a pure connectionist model which learns to compose in a style  
<sup>4</sup> indistinguishable from Bach. To answer this question, we took numerous ideas from deep  
<sup>5</sup> learning research and built a deep LSTM language model for Bach.

<sup>6</sup> While our automatic composition model does surprisingly well in composition tasks, it  
<sup>7</sup> underperforms when asked to harmonize fixed melodies: a task which music theorists consider  
<sup>8</sup> significantly easier than automatic composition.

<sup>9</sup> Both the models the and large-scale evaluation frameworks have been open sourced. The  
<sup>10</sup> models are currently being ported to run on Tensorflow and will be part of the Google Magenta  
<sup>11</sup> project. We plan to continue running the evaluation study on [bachbot.com](#) and will update the  
<sup>12</sup> website as more responses are collected. A press release on SoftwareEngineeringDaily (>10K  
<sup>13</sup> daily users) and a feature on Cambridge University’s front page are both scheduled to go live  
<sup>14</sup> early September, so we expect a significant increase in the amount of responses.

## <sup>15</sup> 8.3 Extensions and Future Work

<sup>16</sup> Looking through [fig. 7.9](#) on page 88, we see that certain samples are virtually indistinguishable  
<sup>17</sup> from Bach while others can be identified more than 80% of the time. It would be valuable to  
<sup>18</sup> understand what features are present in the failure cases which make them easy to distinguish,  
<sup>19</sup> a potential extension for any interested music theorists.

<sup>20</sup> One significant opportunity for improvement is to account for future context during har-  
<sup>21</sup> monization tasks. Specifically, the requirement that our model can be sampled to generate  
<sup>22</sup> compositions constrains its architecture to be unidirectional, significantly impacting its perfor-  
<sup>23</sup> mance on harmonization tasks where future outputs are constrained.

<sup>24</sup> One method to address this is to apply bidirectional RNNs[63] and the sequence to sequence  
<sup>25</sup> framework[126] to map the constrained parts to the harmonized parts while accounting for both  
<sup>26</sup> past and future context. An attention mechanism similar to Bahdanau, Cho, and Bengio [7]  
<sup>27</sup> could be introduced on top of the bidirectional RNN to both enable the model to selectively  
<sup>28</sup> attend to specific time intervals within the context as well as provide insight into what the  
<sup>29</sup> model deems relevant when generating harmonies.

<sup>30</sup> Another way to account for future constraints is using a lookahead search. Instead of gen-  
<sup>31</sup> erating outputs by greedily sampling the RNN’s predictions at each time, the RNN is expanded  
<sup>32</sup> for multiple timesteps and the overall best path is selected. This approach is significantly com-  
<sup>33</sup> plicated by an exponential growth in possible states ( $O(128^L)$  states when looking ahead  $L$

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

Another interesting area for further exploration is in how the different parts are ordered when flattening music scores (where all the notes in a chord are played simultaneously) into encoded token sequences (where a sequential ordering is imposed on the notes in a chord). Harmonization results from section 6.4 on page 76 showed that the Soprano and Alto parts achieved significantly higher error rates than Tenor and Bass parts, which might be attributed to the SATB order imposed when encoding chords. We expect ordering the parts to be harmonized last should improve the model as the fixed parts can now provide additional context aiding more consistent harmony prediction.

We use the Bach chorales because they are fairly homogeneous, widely available in a respectable quantity, and well studied by music theorists. However, our proposed encoding scheme extends to arbitrary degrees of polyphony and musical style. As evidenced by fig. 6.2 on page 77, 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 our model’s generalality and its failure modes by applying the model to other styles of music.

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

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

# References

- [1] **abc:standard [abc wiki]**. July 2015. URL: <http://abcnotation.com/wiki/abc:standard>. 2
- [2] Moray Allan and Christopher KI Williams. “allan2005”. In: **Advances in Neural Information Processing Systems** 17 (2005), pp. 25–32. 3  
4
- [3] Adam Alpern. “Techniques for algorithmic composition of music”. In: **On the web: http://hamp. hampshire. edu/adaF92/algocomp/algocomp** 95 (1995), p. 120. 5  
6
- [4] Charles Ames. “The Markov process as a compositional model: A survey and tutorial”. In: **Leonardo** (1989), pp. 175–187. 7  
8
- [5] Christopher Ariza. “The interrogator as critic: The turing test and the evaluation of generative music systems”. In: **Computer Music Journal** 33.2 (2009), pp. 48–70. 9  
10
- [6] Christopher Ariza and M Cuthbert. **Modeling beats, accents, beams, and time signatures hierarchically with music21 meter objects**. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 2010. 11  
12  
13
- [7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “#4 Neural Machine Translation by Jointly Learning to Align and Translate”. In: (2015), pp. 1–15. arXiv: [1409.0473](https://arxiv.org/pdf/1409.0473v6.pdf). URL: <http://arxiv.org/pdf/1409.0473v6.pdf>. 14  
15  
16
- [8] Justin Bayer et al. “On fast dropout and its applicability to recurrent networks”. In: **arXiv preprint arXiv:1311.0701** (2013). 17  
18
- [9] Matthew I Bellgard and Chi-Ping Tsang. “Harmonizing music the Boltzmann way”. In: **Connection Science** 6.2-3 (1994), pp. 281–297. 19  
20
- [10] Samy Bengio et al. “Scheduled sampling for sequence prediction with recurrent neural networks”. In: **Advances in Neural Information Processing Systems**. 2015, pp. 1171–1179. 21  
22  
23
- [11] Yoshua Bengio. “Learning deep architectures for AI”. In: **Foundations and trends® in Machine Learning** 2.1 (2009), pp. 1–127. 24  
25
- [12] Yoshua Bengio and Olivier Delalleau. “On the expressive power of deep architectures”. In: **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)** 6925 LNAI (2011), pp. 18–36. ISSN: 03029743. DOI: [10.1007/978-3-642-24412-4\\_3](https://doi.org/10.1007/978-3-642-24412-4_3). arXiv: [1206.5533](https://arxiv.org/abs/1206.5533). 26  
27  
28  
29
- [13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning Long-Term Dependencies with Gradient Descent is Difficult”. In: **IEEE Transactions on Neural Networks** 5.2 (1994), pp. 157–166. ISSN: 1045-9227. DOI: [10.1109/72.279181](https://doi.org/10.1109/72.279181). arXiv: [arXiv: 1211.5063v2](https://arxiv.org/abs/1211.5063v2). URL: <http://jmlr.org/proceedings/papers/v28/pascanu13.pdf>. 30  
31  
32  
33
- [14] Jamshed J Bharucha and Peter M Todd. “Modeling the perception of tonal structure with neural nets”. In: **Computer Music Journal** 13.4 (1989), pp. 44–53. 34  
35

- 1 [15] Nicolas Boulanger-Lewandowski, Pascal Vincent, and Yoshua Bengio. “Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription”. In: **Proceedings of the 29th International Conference on Machine Learning (ICML-12)** Cd (2012), pp. 1159–1166. arXiv: [1206.6392](https://arxiv.org/abs/1206.6392).
- 2 [16] Edmund Bowles. “Musicke’s handmaiden: Or technology in the service of the arts”. In: **The computer and music** (1970), pp. 3–20.
- 3 [17] Tim O Brien and Iran Roman. “A Recurrent Neural Network for Musical Structure Processing and Expectation”. In: **CS224d: Deep Learning for Natural Language Processing Final Projects** (2016), pp. 1–9.
- 4 [18] Arthur E Bryson, Walter F Denham, and Stewart E Dreyfus. “Optimal programming problems with inequality constraints”. In: **AIAA journal** 1.11 (1963), pp. 2544–2550.
- 5 [19] John Butt. “Bach-Werke-Verzeichnis”. In: **Notes** 55.4 (1999), pp. 890–893.
- 6 [20] Kyunghyun Cho et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: **arXiv preprint arXiv:1406.1078** (2014).
- 7 [21] Ching-Hua Chuan and Elaine Chew. “A hybrid system for automatic generation of style-specific accompaniment”. In: **Proceedings of the 4th International Joint Workshop on Computational Creativity**. 2007, pp. 57–64.
- 8 [22] Tom Collins et al. “Developing and evaluating computational models of musical style”. In: **Artificial Intelligence for Engineering Design, Analysis and Manufacturing** 30.01 (2016), pp. 16–43.
- 9 [23] Wikimedia Commons. **Comparison of various duple note values**. File:Duple note values comparison.png. 2012. URL: [https://en.wikipedia.org/wiki/File:Duple\\_note\\_values\\_comparison.png](https://en.wikipedia.org/wiki/File:Duple_note_values_comparison.png) (visited on 08/10/2016).
- 10 [24] Darrell Conklin and Ian H Witten. “Multiple viewpoint systems for music prediction”. In: **Journal of New Music Research** 24.1 (1995), pp. 51–73.
- 11 [25] Grosvenor Cooper and Leonard B Meyer. **The rhythmic structure of music**. Vol. 118. University of Chicago Press, 1963.
- 12 [26] David Cope. “Computer modeling of musical intelligence in EMI”. In: **Computer Music Journal** 16.2 (1992), pp. 69–83.
- 13 [27] David Cope. **Computer models of musical creativity**. MIT Press Cambridge, 2005.
- 14 [28] David Cope. “Experiments in Music Intelligence”. In: **Proceedings of the International Computer Music Conference**. 1987.
- 15 [29] David H Cope. **Recombinant music composition algorithm and method of using the same**. US Patent 7,696,426. Apr. 2010.
- 16 [30] Eduardo Coutinho et al. “Computational musicology: An artificial life approach”. In: **2005 portuguese conference on artificial intelligence**. IEEE. 2005, pp. 85–93.
- 17 [31] Pedro P Cruz-Alcázar and Enrique Vidal-Ruiz. “Learning regular grammars to model musical style: Comparing different coding schemes”. In: **International Colloquium on Grammatical Inference**. Springer. 1998, pp. 211–222.
- 18 [32] Michael Scott Cuthbert and Christopher Ariza. “music21: A toolkit for computer-aided musicology and symbolic music data”. In: (2010).

- [33] Michael Scott Cuthbert et al. “Hidden Beyond MIDI’s Reach : Feature Extraction and Machine Learning with Rich Symbolic Formats in music21”. In: **Proceedings of the Neural Information Processing Systems Conference** (2011), pp. 3–4. 1  
2  
3
- [34] George Cybenko. “Degree of approximation by superpositions of a sigmoidal function”. In: **Approximation Theory and its Applications** 9.3 (1993), pp. 17–28. ISSN: 10009221. DOI: [10.1007/BF02836480](https://doi.org/10.1007/BF02836480). 4  
5  
6
- [35] Roger B Dannenberg, Belinda Thom, and David Watson. “A machine learning approach to musical style recognition”. In: (1997). 7  
8
- [36] James Denny. **The Oxford school harmony course**. Vol. 1. Oxford University Press, 1960. 9  
10
- [37] Christine Denton and M Fillion. “The History of Musical Tuning and Temperament during the Classical and Romantic Periods”. In: (1997). 11  
12
- [38] Jacob Devlin et al. “Fast and Robust Neural Network Joint Models for Statistical Machine Translation.” In: **ACL (1)**. Citeseer. 2014, pp. 1370–1380. 13  
14
- [39] Mark Dolson. “Machine tongues XII: Neural networks”. In: **Computer Music Journal** 13.3 (1989), pp. 28–40. 15  
16
- [40] Julie S Downs et al. “Are your participants gaming the system?: screening mechanical turk workers”. In: **Proceedings of the SIGCHI Conference on Human Factors in Computing Systems**. ACM. 2010, pp. 2399–2402. 17  
18  
19
- [41] Kemal Ebcioğlu. “An expert system for harmonizing four-part chorales”. In: **Computer Music Journal** 12.3 (1988), pp. 43–51. 20  
21
- [42] D. Eck and J. Schmidhuber. “Finding temporal structure in music: Blues improvisation with LSTM recurrent networks”. In: **Neural Networks for Signal Processing - Proceedings of the IEEE Workshop** 2002-Janua (2002), pp. 747–756. ISSN: 0780376161. DOI: [10.1109/NNSP.2002.1030094](https://doi.org/10.1109/NNSP.2002.1030094). 22  
23  
24  
25
- [43] Douglas Eck and Jasmin Lapalme. “Learning musical structure directly from sequences of music”. In: **University of Montreal, Department of Computer Science, CP 6128** (2008). 26  
27  
28
- [44] Douglas Eck and Jürgen Schmidhuber. “A First Look at Music Composition using LSTM Recurrent Neural Networks”. In: **Idsa** (2002). URL: <http://www.idsia.ch/%7B~%7Djuergen/blues/IDSIA-07-02.pdf>. 29  
30  
31
- [45] Salah El Hihi and Yoshua Bengio. “Hierarchical Recurrent Neural Networks for Long-Term Dependencies.” In: **NIPS**. Vol. 400. Citeseer. 1995, p. 409. 32  
33
- [46] Jeffrey L Elman. “Finding structure in time”. In: **Cognitive science** 14.2 (1990), pp. 179–211. 34  
35
- [47] Johannes Feulner and Dominik Hörnel. “Melonet: Neural networks that learn harmony-based melodic variations”. In: **Proceedings of the International Computer Music Conference**. INTERNATIONAL COMPUTER MUSIC ACCOCIATION. 1994, pp. 121–121. 36  
37  
39
- [48] JA Franklin et al. “Learning and improvisation”. In: **Neural Information Processing Systems**. Vol. 14. 2001. 40  
41
- [49] Judy A Franklin. “Jazz Melody Generation from Recurrent Network Learning of Several Human Melodies.” In: **FLAIRS Conference**. 2005, pp. 57–62. 42  
43

- 1 [50] Judy A Franklin. “Predicting reinforcement of pitch sequences via lstm and td”. In: **Proc. of International Computer Music Conference, Miami, Florida.** Vol. 306. 2004.
- 2 [51] Judy A Franklin. “Recurrent Neural Networks and Pitch Representations for Music 3 Tasks.” In: **FLAIRS Conference.** 2004, pp. 33–37.
- 6 [52] Judy A Franklin. “Recurrent neural networks for music computation”. In: **INFORMS 7 Journal on Computing** 18.3 (2006), pp. 321–338.
- 8 [53] Dylan Freedman. “Correlational Harmonic Metrics: Bridging Computational and Human 9 Notions of Musical Harmony”. PhD thesis. 2015.
- 10 [54] Felix A Gers and E Schmidhuber. “LSTM recurrent networks learn simple context-free 11 and context-sensitive languages”. In: **IEEE Transactions on Neural Networks** 12.6 12 (2001), pp. 1333–1340.
- 13 [55] Felix A Gers and Jürgen Schmidhuber. “Recurrent nets that time and count”. In: **Neural 14 Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International 15 Joint Conference on.** Vol. 3. IEEE. 2000, pp. 189–194.
- 16 [56] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. “Learning to forget: Continual 17 prediction with LSTM”. In: **Neural computation** 12.10 (2000), pp. 2451–2471.
- 18 [57] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. “Learning precise timing 19 with LSTM recurrent networks”. In: **Journal of machine learning research** 3.Aug 20 (2002), pp. 115–143.
- 21 [58] Felix A Gers et al. “DEKF-LSTM.” In: **ESANN.** 2002, pp. 369–376.
- 22 [59] Kratarth Goel, Raunaq Vohra, and JK Sahoo. “Polyphonic music generation by 23 modeling temporal dependencies using a RNN-DBN”. In: **International Conference on 24 Artificial Neural Networks.** Springer. 2014, pp. 217–224.
- 25 [60] Christoph Goller and Andreas Kuchler. “Learning task-dependent distributed representations 26 by backpropagation through structure”. In: **Neural Networks, 1996., IEEE 27 International Conference on.** Vol. 1. IEEE. 1996, pp. 347–352.
- 28 [61] Ian Goodfellow et al. “Generative adversarial nets”. In: **Advances in Neural Information 29 Processing Systems.** 2014, pp. 2672–2680.
- 30 [62] Alex Graves. “Generating sequences with recurrent neural networks”. In: **arXiv preprint arXiv:1308.0850** (2013).
- 32 [63] Alex Graves and J??rgen Schmidhuber. “Framewise phoneme classification with bi- 33 directional LSTM networks”. In: **Proceedings of the International Joint Conference 34 on Neural Networks** 4 (2005), pp. 2047–2052. ISSN: 08936080. DOI: [10.1109/IJCNN.2005.1556215](https://doi.org/10.1109/IJCNN.2005.1556215).
- 36 [64] Klaus Greff et al. “LSTM: A search space odyssey”. In: **arXiv preprint arXiv:1503.04069** 37 (2015).
- 38 [65] Niall Griffith and Peter M Todd. **Musical networks: Parallel distributed perception 39 and performance.** MIT Press, 1999.
- 40 [66] PDP Research Group et al. “Parallel distributed processing: Explorations in the mi- 41 crostructure of cognition: Vol. 1”. In: **Foundations.** Cambridge, MA: MIT Press 42 (1986).

- 
- [67] Stephen Handel. **Listening: An introduction to the perception of auditory events.** The MIT Press, 1993. 1  
2
- [68] Kenneth Heafield et al. “Scalable Modified Kneser-Ney Language Model Estimation”. 3  
In: **Proceedings of the 51st Annual Meeting of the Association for Computational**  
**Linguistics.** Sofia, Bulgaria, Aug. 2013. URL: [http://kheafield.com/professional/edinburgh/estimate%5C\\_paper.pdf](http://kheafield.com/professional/edinburgh/estimate%5C_paper.pdf). 4  
5  
6
- [69] William Herlands et al. “A Machine Learning Approach to Musically Meaningful Ho- 7  
mogeneous Style Classification”. In: **Twenty-Eighth AAAI Conference on Artificial**  
**Intelligence** (2014), pp. 276–282. URL: <http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8314>. 8  
9  
10
- [70] Hermann Hild, Johannes Feulner, and Wolfram Menzel. “HARMONET: A neural net 11  
for harmonizing chorales in the style of JS Bach”. In: **NIPS.** 1991, pp. 267–274. 12
- [71] Geoffrey E Hinton and Terrence J Sejnowski. “Learning and relearning in Boltzmann 13  
machines”. In: **Parallel distributed processing: Explorations in the microstructure**  
**of cognition** 1 (1986), pp. 282–317. 14  
15
- [72] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: **Neural**  
**computation** 9.8 (1997), pp. 1735–1780. 16  
17
- [73] Peter Hoffmann. “Towards an” automated art”: Algorithmic processes in Xenakis’ 18  
compositions”. In: **Contemporary Music Review** 21.2-3 (2002), pp. 121–131. 19
- [74] Dominik Hörmel. “MELONET I: Neural nets for inventing baroque-style chorale vari- 20  
ations”. In: **NIPS.** 1997, pp. 887–893. 21
- [75] Dominik Hornel and Tomas Ragg. “Learning Musical Structure and Style by Recog- 22  
nition, Prediction and Evolution”. In: (1996). 23
- [76] Brian Hyer. **Tonality.** URL: <http://www.oxfordmusiconline.com/subscriber/article/grove/music/28102> (visited on 08/10/2016). 24  
25
- [77] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network 26  
training by reducing internal covariate shift”. In: **arXiv preprint arXiv:1502.03167** 27  
(2015). 28
- [78] Oswald Jonas and John Rothgeb. **Introduction to the theory of Heinrich Schenker:**  
**the nature of the musical work of art.** New York: Longman, 1982. 29  
30
- [79] Michael I Jordan. “Serial order: A parallel distributed processing approach”. In: **Ad-**  
**vances in psychology** 121 (1997), pp. 471–495. 31  
32
- [80] Łukasz Kaiser and Ilya Sutskever. “Neural gpus learn algorithms”. In: **arXiv preprint**  
**arXiv:1511.08228** (2015). 33  
34
- [81] Barry Kernfeld. **Meter.** URL: [www.oxfordmusiconline.com/subscriber/article\\_citations/grove/music/J298700](http://www.oxfordmusiconline.com/subscriber/article_citations/grove/music/J298700) (visited on 08/10/2016). 35  
36
- [82] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: **arXiv preprint arXiv:1412.6980** (2014). 37  
38
- [83] Jan Koutnik et al. “A Clockwork RNN”. In: **Proceedings of The 31st International**  
**Conference on Machine Learning** 32 (2014), pp. 1863–1871. arXiv: [arXiv:1402.3511v1](http://arxiv.org/abs/1402.3511v1). URL: <http://jmlr.org/proceedings/papers/v32/koutnik14.html>. 39  
40  
41

- [84] Carol L Krumhansl. **Cognitive foundations of musical pitch**. Oxford University Press, 2001.
- [85] Bernice Laden and Douglas H Keefe. “The representation of pitch in a neural net model of chord classification”. In: **Computer Music Journal** 13.4 (1989), pp. 12–26.
- [86] Fred Lerdahl. **Jackendoff. A Generative Theory of Tonal Music**. 1983.
- [87] Rensis Likert. “A technique for the measurement of attitudes.” In: **Archives of psychology** (1932).
- [88] Seppo Linnainmaa. “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors”. In: **Master’s Thesis (in Finnish)**, Univ. Helsinki (1970), pp. 6–7.
- [89] I-Ting Liu and Bhiksha Ramakrishnan. “Bach in 2014: Music Composition with Recurrent Neural Network”. In: **arXiv:1412.3191** 5 (2014), pp. 1–9. arXiv: [1412.3191](https://arxiv.org/abs/1412.3191). URL: <http://arxiv.org/abs/1412.3191>.
- [90] Xunying Liu et al. “Efficient lattice resoring using recurrent neural network language models”. In: **2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. IEEE. 2014, pp. 4908–4912.
- [91] Qi Lyu. “Polyphonic Music Modelling with LSTM-RTRBM”. In: **Proceedings of the 23rd Annual ACM Conference on Multimedia Conference** (2015), pp. 991–994.
- [92] Michael I Mandel, Graham E Poliner, and Daniel PW Ellis. “Support vector machine active learning for music retrieval”. In: **Multimedia systems** 12.1 (2006), pp. 3–13.
- [93] Tomáš Mikolov. “Statistical Language Models Based on Neural Networks”. PhD thesis. Brno, CZ, 2012, p. 129. URL: [http://www.fit.vutbr.cz/research/view\\_pub.php?id=10158](http://www.fit.vutbr.cz/research/view_pub.php?id=10158).
- [94] Tomas Mikolov et al. “Learning Longer Memory in Recurrent Neural Networks”. In: **Iclr** (2015), pp. 1–9. arXiv: [arXiv:1412.7753v1](https://arxiv.org/abs/1412.7753v1). URL: <http://arxiv.org/pdf/1412.7753v1.pdf>.
- [95] T Mikolov et al. “Recurrent Neural Network based Language Model”. In: **Interspeech** September (2010), pp. 1045–1048.
- [96] Michael C Mozer. “Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing”. In: **Connection Science** 6.2-3 (1994), pp. 247–280.
- [97] Dylan Jeremy Nagler. “SCHUBOT: Machine Learning Tools for the Automated Analysis of Schubert’s Lieder”. PhD thesis. 2014.
- [98] Jean-Jacques Nattiez. **Music and discourse: Toward a semiology of music**. Princeton University Press, 1990.
- [99] Aran Nayebi and Matt Vitelli. “GRUV: Algorithmic Music Generation using Recurrent Neural Networks”. In: **CS224d: Deep Learning for Natural Language Processing Final Projects** (2015), pp. 1–6.
- [100] Yizhao Ni et al. “An end-to-end machine learning system for harmonic analysis of music”. In: **IEEE Transactions on Audio, Speech, and Language Processing** 20.6 (2012), pp. 1771–1783.

- [101] Peter Norvig. **Paradigms of artificial intelligence programming: case studies in Common LISP**. Morgan Kaufmann, 1992.
- [102] **Online Research Panel Pricing | SurveyMonkey Audience**. URL: <https://www.surveymonkey.co.uk/mp/audience/audience-pricing/>.
- [103] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks”. In: **Proceedings of The 30th International Conference on Machine Learning** 2 (2012), pp. 1310–1318. ISSN: 1045-9227. DOI: [10.1109/72.279181](https://doi.org/10.1109/72.279181). arXiv: [arXiv:1211.5063v2](https://arxiv.org/abs/1211.5063v2). URL: <http://jmlr.org/proceedings/papers/v28/pascanu13.pdf>.
- [104] Razvan Pascanu et al. “How to construct deep recurrent neural networks”. In: **arXiv preprint arXiv:1312.6026** (2013).
- [105] Marcus Pearce, David Meredith, and Geraint Wiggins. “Motivations and methodologies for automation of the compositional process”. In: **Musicae Scientiae** 6.2 (2002), pp. 119–147.
- [106] Marcus Pearce and Geraint Wiggins. “Towards a framework for the evaluation of machine compositions”. In: **Proceedings of the AISB’01 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences**. Citeseer. 2001, pp. 22–32.
- [107] Walter Piston. “Harmony. (Revised and expanded by Mark DeVoto)”. In: **Londres: Victor Gollancz LTD** (1978).
- [108] Donya Quick. “Kulitta: A Framework for Automated Music Composition”. PhD thesis. YALE UNIVERSITY, 2014.
- [109] Donya Quick and Paul Hudak. **A temporal generative graph grammar for harmonic and metrical structure**. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 2013.
- [110] Daniel Ramage. “Hidden Markov models fundamentals”. In: **Lecture Notes**. <http://cs229.stanford.edu/section/cs229-hmm.pdf> (2007).
- [111] Don Michael Randel. **The Harvard concise dictionary of music and musicians**. Harvard University Press, 1999.
- [112] Martin Riedmiller and Heinrich Braun. “A direct adaptive method for faster backpropagation learning: The RPROP algorithm”. In: **Neural Networks, 1993., IEEE International Conference On**. IEEE. 1993, pp. 586–591.
- [113] AJ Robinson and Frank Fallside. **The utility driven dynamic error propagation network**. University of Cambridge Department of Engineering, 1987.
- [114] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: **Cognitive modeling** 5.3 (1988), p. 1.
- [115] Heinrich Schenker. “Harmony, ed. Oswald Jonas, trans. Elisabeth Mann Borgese”. In: **Chicago: University of Chicago Press** 35 (1954), pp. 105–32.
- [116] Jürgen Schmidhuber. “Learning complex, extended sequences using the principle of history compression”. In: **Neural Computation** 4.2 (1992), pp. 234–242.
- [117] Marco Scirea et al. “MetaCompose: A Compositional Evolutionary Music Composer”. In: **International Conference on Evolutionary and Biologically Inspired Music and Art**. Springer. 2016, pp. 202–217.

- [118] Roger N Shepard. “Geometrical approximations to the structure of musical pitch.” In: **Psychological review** 89.4 (1982), p. 305.
- [119] Randall R Spangler, Rodney M Goodman, and Jim Hawkins. “Bach in a Box-Real-Time Harmony”. In: (1998).
- [120] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting.” In: **Journal of Machine Learning Research** 15.1 (2014), pp. 1929–1958.
- [121] Efstathios Stamatatos and Gerhard Widmer. “Automatic identification of music performers with learning ensembles”. In: **Artificial Intelligence** 165.1 (2005), pp. 37–56.
- [122] Andreas Stolcke et al. “SRILM-an extensible language modeling toolkit.” In: **Interspeech**. Vol. 2002. 2002, p. 2002.
- [123] Bob L Sturm et al. “Music transcription modelling and composition using deep learning”. In: **arXiv preprint arXiv:1604.08723** (2016).
- [124] Bob Sturm, Joao Felipe Santos, and Iryna Korshunova. “Folk music style modelling by recurrent neural networks with long short term memory units”. In: **16th International Society for Music Information Retrieval Conference**. 2015.
- [125] Ilya Sutskever. “Training Recurrent Neural Networks - Ilia Sutskever - PhD thesis”. In: (). URL: [http://www.cs.utoronto.ca/%5C%7Eilya/pubs/ilya%5C\\_sutskever%5C\\_phd%5C\\_thesis.pdf](http://www.cs.utoronto.ca/%5C%7Eilya/pubs/ilya%5C_sutskever%5C_phd%5C_thesis.pdf).
- [126] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to sequence learning with neural networks”. In: **Advances in neural information processing systems**. 2014, pp. 3104–3112.
- [127] Richard S Sutton and Andrew G Barto. **Reinforcement learning: An introduction**. Vol. 1. 1. MIT press Cambridge, 1998.
- [128] Ernst Terhardt. “Pitch, consonance, and harmony”. In: **The Journal of the Acoustical Society of America** 55.5 (1974), pp. 1061–1069. DOI: <http://dx.doi.org/10.1121/1.1914648>. URL: <http://scitation.aip.org/content/asa/journal/jasa/55/5/10.1121/1.1914648>.
- [129] Peter Todd. “A sequential network design for musical applications”. In: **Proceedings of the 1988 connectionist models summer school**. 1988, pp. 76–84.
- [130] Peter M Todd. “A connectionist approach to algorithmic composition”. In: **Computer Music Journal** 13.4 (1989), pp. 27–43.
- [131] Petri Toiviainen. **Symbolic AI versus Connectionism in Music Research**. 2000.
- [132] Chi Ping Tsang and Melanie Aitken. “Harmonizing Music as a Discipline in Constraint Logic Programming”. In: **Proceedings of the International Computer Music Conference**. INTERNATIONAL COMPUTER MUSIC ACCOCIATION. 1991, pp. 61–61.
- [133] Alan M Turing. “Computing machinery and intelligence”. In: **Mind** 59.236 (1950), pp. 433–460.
- [134] Dmitri Tymoczko. “Three conceptions of musical distance”. In: **Communications in Computer and Information Science** 38 (2009), pp. 258–272. ISSN: 18650929. DOI: [10.1007/978-3-642-02394-1\\_24](10.1007/978-3-642-02394-1_24).

- [135] Eric Wanner. “The ATN and the sausage machine: Which one is baloney?” In: **Cognition** 8.2 (1980), pp. 209–225. 1  
2
- [136] Gil Weinberg et al. “A real-time genetic algorithm in human-robot musical improvisation”. In: **International Symposium on Computer Music Modeling and Retrieval**. Springer. 2007, pp. 351–359. 3  
4  
5
- [137] John David White and William E Lake. **Guidelines for college teaching of music theory**. Scarecrow Press, 2002. 6  
7
- [138] Ronald J Williams and Jing Peng. “An efficient gradient-based algorithm for on-line training of recurrent network trajectories”. In: **Neural computation** 2.4 (1990), pp. 490–501. 8  
9  
10
- [139] Ronald J Williams and David Zipser. “A learning algorithm for continually running fully recurrent neural networks”. In: **Neural computation** 1.2 (1989), pp. 270–280. 11  
12
- [140] Ronald J Williams and David Zipser. “Gradient-based learning algorithms for recurrent networks and their computational complexity”. In: **Back-propagation: Theory, architectures and applications** (1995), pp. 433–486. 13  
14  
15
- [141] Terry Winograd. “Linguistics and the computer analysis of tonal harmony”. In: **Journal of Music Theory** 12.1 (1968), pp. 2–49. 16  
17
- [142] Kevin B. Wright. “Researching Internet-Based Populations: Advantages and Disadvantages of Online Survey Research, Online Questionnaire Authoring Software Packages, and Web Survey Services”. In: **Journal of Computer-Mediated Communication** 10.3 (2005), pp. 00–00. ISSN: 1083-6101. DOI: [10.1111/j.1083-6101.2005.tb00259.x](https://doi.org/10.1111/j.1083-6101.2005.tb00259.x). URL: <http://dx.doi.org/10.1111/j.1083-6101.2005.tb00259.x>. 18  
19  
20  
21  
22
- [143] Wojciech Zaremba. “An empirical exploration of recurrent network architectures”. In: (2015). 23  
24
- [144] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. “Recurrent neural network regularization”. In: **arXiv preprint arXiv:1409.2329** (2014). 25  
26

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

# A

## Appendix A

### A.1 A primer on Western music theory

This section provides a primer on the music theory relevant for understanding the following chapters. It synthesizes material originally presented in Franklin [52], Nagler [97], Quick [108], and Freedman [53]. Readers with a strong music background may wish to skip to ?? on page ??.

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

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



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

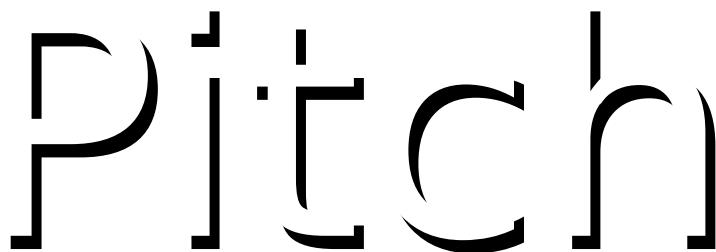


Fig. A.2 Terhardt's visual analogy for pitch. Similar to how the viewer of this figure may perceive contours not present, pitch describes subjective information received by the listener even when physical frequencies are absent.

### <sup>1</sup> A.1.1 Notes: the basic building blocks

<sup>2</sup> A **note** is the most fundamental element of music score and represents a sound played at a  
<sup>3</sup> certain **pitch** for a certain **duration**. In sheet music such as [fig. A.1](#), the notes are denoted by  
<sup>4</sup> the filled/unfilled black heads with protruding stems. As a can be viewed as a collection of  
<sup>5</sup> notes over time, notes are the fundamental building blocks for musical scores.

### <sup>6</sup> Pitch

<sup>7</sup> Though pitch is closely related to physical frequency of vibration of a waveform (as measured  
<sup>8</sup> in Hertz), pitch is a perceptual property whose semantic meaning is derived from a listener's  
<sup>9</sup> perception. This distinction has been scrutinized by Terhardt [128], whose visual analogy in  
<sup>10</sup> [fig. A.2](#) illustrates how a pitch can be heard even if its perceived frequency is absent just as one  
<sup>11</sup> may see the word "PITCH" despite being presented with only a suggestive shadow.

<sup>12</sup> Despite its psychoacoustic nature, it is nevertheless useful to objectively quantify pitch as  
<sup>13</sup> a frequency. To do so, we first need some definitions. The difference between two frequencies  
<sup>14</sup> is called an **interval** and an **octave** is an interval corresponding to the distance between a

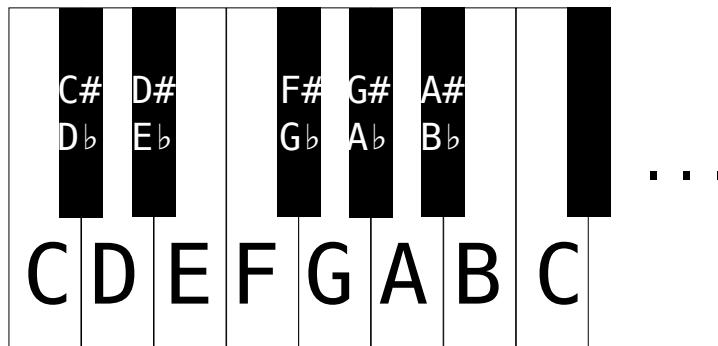


Fig. A.3 Illustration of an octave in the 12-note chromatic scale on a piano keyboard.

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

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

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

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

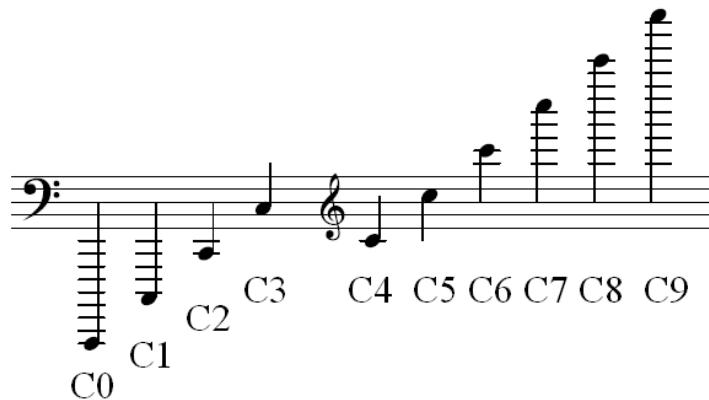


Fig. A.4 Scientific pitch notation and sheet music notation of *C* notes at ten different octaves.

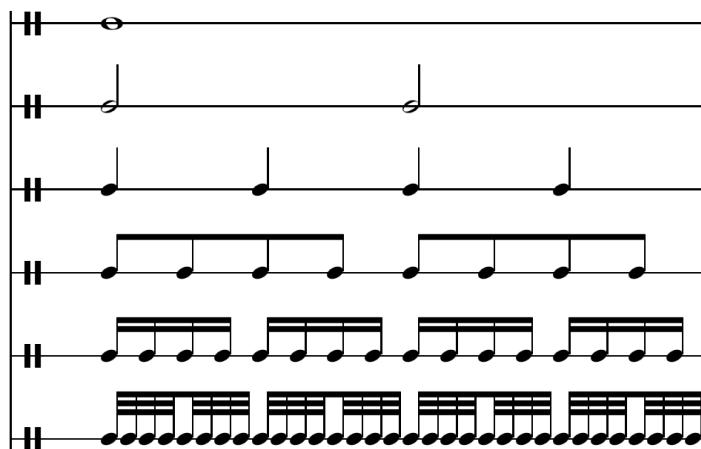


Fig. A.5 Comparison of various note durations [23]

## <sup>1</sup> Duration

- <sup>2</sup> In addition to pitch, a note also possesses a **duration**. The duration of a note indicates how
- <sup>3</sup> long it is to be played and is measured in fractions of a **whole note** (American) or **semi-**
- <sup>4</sup> **breve** (British). Perhaps the most common duration is a **quarter-note** (American) or **crotchet**
- <sup>5</sup> (British). Other note durations are also possible and the most common along with their nota-
- <sup>6</sup> tion in sheet music are enumerated in fig. A.5. The relationship between durations and physical
- <sup>7</sup> time intervals is given by the **tempo**, which is usually denoted near the start of the piece in beats
- <sup>8</sup> per minute.

**Offset, Measures, and Meter**

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

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

In [fig. A.1](#) on page 106, the crotchet preceding the first bar provides an example of an anacrusis. Notice that all other measures in the score are four crotchets in duration. In addition, observe that the offsets of notes within a measure is highly repetitive. There is always a note articulated on the first crotchet of a measure and articulations occurring between crotchets (*i.e.* quavers) are only present the last two crotchets of a measure.

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

**Piano roll notation**

[Figure A.6](#) shows the the same score from [fig. A.1](#) on page 106 in **piano roll notation**, a format which is convenient for visualization purposes. The horizontal and vertical axes represent time and pitch respectively. A solid horizontal bar signifies the presence of a note at the corresponding pitch and offset and the length of the bar corresponds to the note's duration.

**A.1.2 Tonality in common practice music**

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

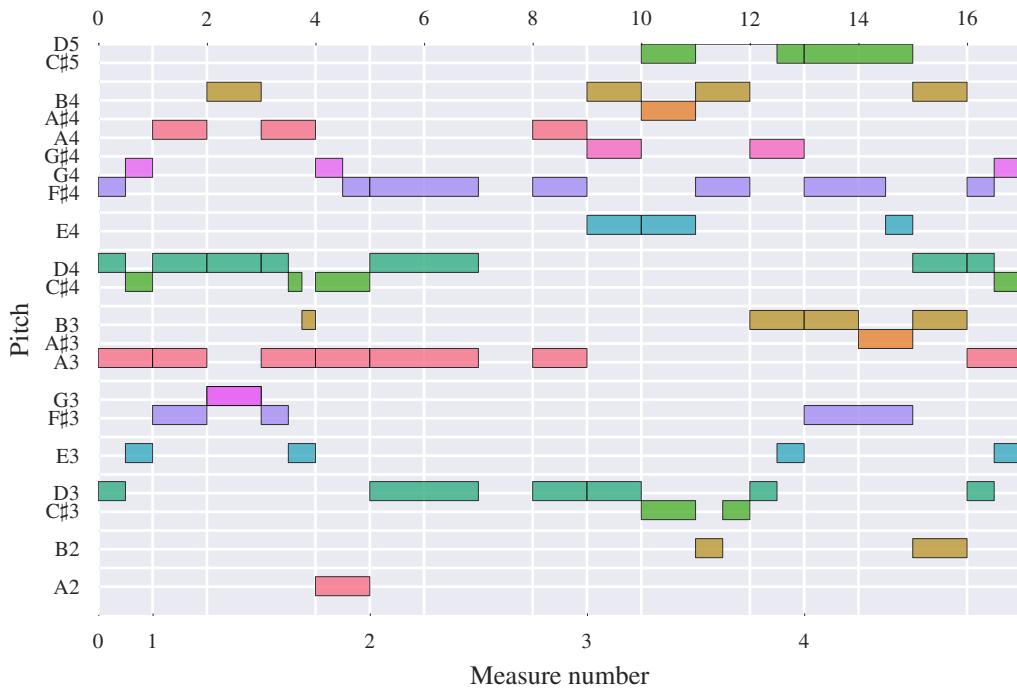


Fig. A.6 Piano roll notation of the music in [fig. A.1](#)

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

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

### <sup>1</sup> A.1.3 Polyphony, chords, and chord progressions

<sup>2</sup> Whereas **monophonic** music is characterized by the presence of a single **part** sounding at most  
<sup>3</sup> one note at any given time, **polyphonic** music contains multiple parts potentially sounding  
<sup>4</sup> multiple pitches at the same time. Just as notes form the basis of monophonic music, chords  
<sup>5</sup> are the fundamental building blocks for polyphonic music.

### <sup>6</sup> Chords: basic units for representing simultaneously sounding notes

<sup>7</sup> A **chord** is a collection of three or more pitches all sounding simultaneously [111]. In Western  
<sup>8</sup> classical music, they typically consist of a **root note** whose pitch class forms a base from  
<sup>9</sup> which successive notes are built upon. The intervals between the pitch classes in a chord are

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

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

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

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

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

### Chord progressions, phrases, and cadences

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

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

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

5 fliang: Review that these definitions are actually necessary

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

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

12 **Plagal cadence** : IV–I

13 **Interrupted cadence** : V–vi. Also considered a weak cadence which invokes a “hanging”  
14 sensation prompting continuation [107].

15 **Transposition invariance**

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

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

25 **A.2 An introduction to neural networks**

26 **A.2.1 Neurons: the basic computation unit**

27 Neurons are the basic abstraction which are combined together to form neural networks. A  
28 **neuron** is a parametric model of a function  $f : \mathbb{R}^D \rightarrow \mathbb{R}$  from its  $D$ -dimensional input  $\mathbf{x}$  to

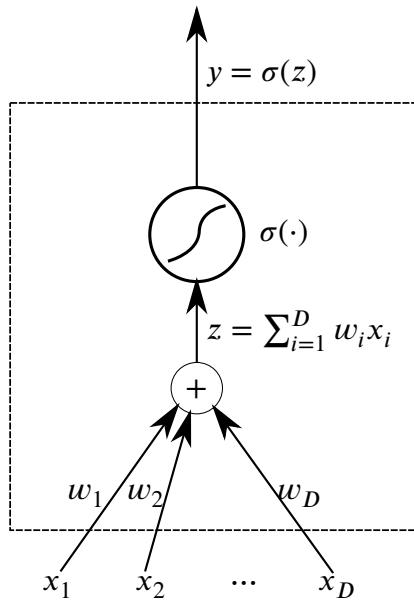


Fig. A.7 A single neuron first computes an activation  $z$  and then passes it through an activation function  $\sigma(\cdot)$

its output  $y$ . Our neurons will be defined as

$$f(\mathbf{x}) := \sigma(\langle \mathbf{w}, \mathbf{x} \rangle) \quad (\text{A.1})$$

which can be viewed as an inner product with **weights**  $\mathbf{w}$  to produce an **activation**  $z := \langle \mathbf{w}, \mathbf{x} \rangle \in \mathbb{R}$  which is then squashed to a bounded domain by a non-linear **activation function**  $\sigma : \mathbb{R} \rightarrow [L, U]$ . This is visually depicted in fig. A.7, which also makes apparent the interpretation of weight  $w_i$  as the sensitivity of the output  $y$  to the input  $x_i$ .

### A.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**.

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

The improved modeling capacity gained by composing multiple layers is due to the composition of multiple non-linear activation functions. In fact, it is easy to show that removing activation functions would make a deep network equivalent to a single matrix transform: let

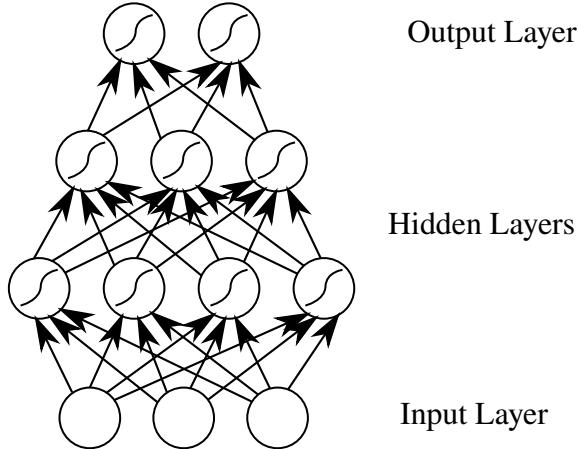


Fig. A.8 Graph depiction of a feedforward neural network with 2 hidden layers

<sup>1</sup>  $\mathbf{W}_{l,l+1}$  denote the weights between layers  $l$  and  $l + 1$ . The original neural network computes  
<sup>2</sup> the function

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

<sup>4</sup> After removing the activation functions  $\sigma$ , we are left with

$$\mathbf{W}_{L,L-1}\mathbf{W}_{L-1,L-2}\cdots\mathbf{W}_{2,1}\mathbf{x} = \mathbf{x} = \tilde{\mathbf{W}}\mathbf{x} \quad (\text{A.3})$$

<sup>6</sup> where  $\tilde{\mathbf{W}} = (\prod_{i=1}^{L-1} \mathbf{W}_{i,i+1})$  is a matrix transform computing the same function as the neural  
<sup>7</sup> network with activation functions removed.

### <sup>8</sup> A.2.3 Recurrent neural networks

<sup>9</sup> While feedforward neural networks provide a flexible model for approximating arbitrary func-  
<sup>10</sup> tions, they require a fixed-dimension input  $\mathbf{x}$  and hence cannot be directly applied to sequential  
<sup>11</sup> data  $\mathbf{x} = (\mathbf{x}_t)_{t=1}^T$  where  $T$  may vary.

<sup>12</sup> A naive method for extending feedforward networks would be to independently apply a  
<sup>13</sup> feedforward network to compute  $\mathbf{y}_t = f(\mathbf{x}_t \theta)$  at each timestep  $1 \leq t \leq T$ . However, this  
<sup>14</sup> approach is only correct when each output  $\mathbf{y}_t$  depends only on the input at the current time  $\mathbf{x}_t$   
<sup>15</sup> and is independent of all prior inputs  $\{\mathbf{x}_k\}_{k < t}$ . This assumption is false in musical data: the  
<sup>16</sup> current musical note usually is highly dependent on the sequence of notes leading up to it.

<sup>17</sup> This shortcoming motivates **recurrent neural networks** (RNNs), which generalize feed-  
<sup>18</sup> forward networks by introducing time-delayed recurrent connections between hidden layers  
<sup>19</sup> (Elman networks [46]) or from the output layers to the hidden layers (Jordan networks [79]).  
<sup>20</sup> Mathematically, an (Elman-type) RNN is a discrete time dynamical system commonly param-

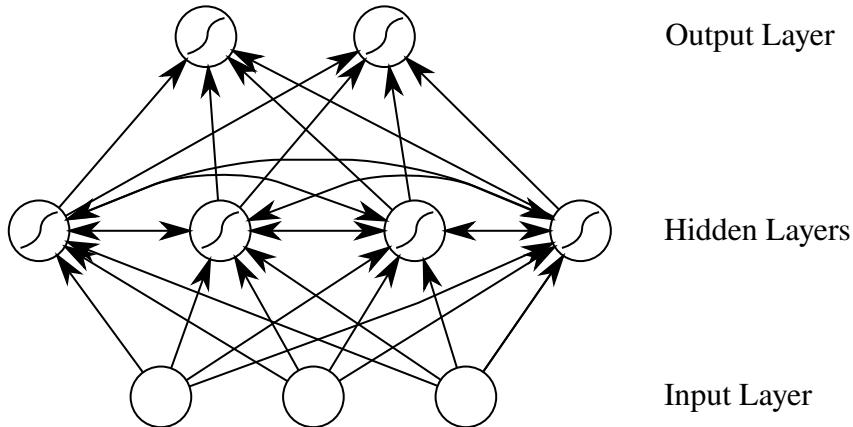


Fig. A.9 Graph representation of an Elman-type RNN.

eterized as:

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

(A.5)

where  $\sigma_{..}(\cdot)$  are activation functions acting element-wise and  $\theta = \{\mathbf{W}_{xh}, \mathbf{W}_{hh}, \mathbf{W}_{hy}\}$  are the learned parameters. fig. A.9 provides a graphical illustration of such a network. Notice that apart from the edges between hidden nodes, the network is identical to a regular feedforward network (fig. A.8).

fliang: Why do we use Elman

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

### A.3 Sufficient conditions for vanishing gradients

Following Pascanu, Mikolov, and Bengio [103], let  $\|\cdot\|$  be any submultiplicative matrix norm (*e.g.* Frobenius, spectral, nuclear, Shatten  $p$ -norms). Without loss of generality, we will use

1 the **operator norm** defined as

$$2 \quad \|A\| = \sup_{x \in \mathbb{R}^n; x \neq 0} \frac{|Ax|}{|x|} \quad (\text{A.6})$$

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

4 Applying the definition of submultiplicativity to the factors of the product in [eq. \(2.4\)](#), we  
5 have that for any  $k$

$$6 \quad \left\| \frac{\partial \mathbf{h}_k}{\partial \mathbf{h}_{k-1}} \right\| \leq \|\mathbf{W}_{hh}^\top\| \|\text{diag}(\sigma'_{hh}(\mathbf{h}_{k-1}))\| \leq \gamma_{\mathbf{W}} \gamma_\sigma \quad (\text{A.7})$$

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

$$8 \quad \gamma_\sigma := \sup_{h \in \mathbb{R}^n} \|\text{diag}(\sigma'_{hh}(\mathbf{h}))\| \quad (\text{A.8})$$

$$9 \quad = \sup_{h \in \mathbb{R}^n} \max_i \sigma'_{hh}(\mathbf{h})_i \quad \text{Operator norm of diag} \quad (\text{A.9})$$

$$10 \quad = \sup_{x \in \mathbb{R}} \sigma'_{hh}(x) \quad \sigma_{hh} \text{ acts elementwise} \quad (\text{A.10})$$

12 Substituting back into [eq. \(2.4\)](#), we find that

$$13 \quad \left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \right\| = \left\| \prod_{t \geq i > k} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \right\| \leq \prod_{t \geq i > k} \left\| \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \right\| \leq (\gamma_{\mathbf{W}} \gamma_\sigma)^{t-k} \quad (\text{A.11})$$

14 Hence, we see that a sufficient condition for vanishing gradients is for  $\gamma_{\mathbf{W}} \gamma_\sigma < 1$ , in which

15 case  $\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \right\| \rightarrow 0$  exponentially for long timespans  $t \gg k$ .

16 If  $\gamma_\sigma$  is bounded, sufficient conditions for vanishing gradients to occur may be written as

$$17 \quad \gamma_{\mathbf{W}} < \frac{1}{\gamma_\sigma} \quad (\text{A.12})$$

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

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

# B

## Appendix B

### B.1 Additional Figures and Tables

Fig. B.1 Results of grid search (see [Section 4.2.5](#)) 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

---

Continued on next page

---

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
2.0	256.0	256.0	32.0	0.307188	0.487868
2.0	256.0	256.0	32.0	0.400955	0.489320
3.0	256.0	256.0	64.0	0.381868	0.489810
2.0	256.0	256.0	64.0	0.333356	0.491396
2.0	256.0	256.0	64.0	0.284248	0.491593
3.0	128.0	128.0	32.0	0.365171	0.492478
3.0	256.0	128.0	32.0	0.264723	0.492849
3.0	384.0	128.0	32.0	0.228556	0.495991
3.0	256.0	128.0	64.0	0.248987	0.496190
3.0	256.0	128.0	32.0	0.445840	0.498205
3.0	256.0	256.0	32.0	0.273567	0.499422
2.0	256.0	128.0	64.0	0.256022	0.500500
3.0	256.0	256.0	32.0	0.338776	0.501711
2.0	128.0	128.0	32.0	0.384075	0.501840
3.0	128.0	128.0	64.0	0.417780	0.501919
2.0	256.0	128.0	32.0	0.219939	0.502503
3.0	128.0	128.0	64.0	0.361381	0.503206
3.0	128.0	128.0	32.0	0.431771	0.503590
3.0	256.0	64.0	64.0	0.263001	0.503945
3.0	256.0	384.0	64.0	0.419091	0.504249
3.0	256.0	256.0	32.0	0.393463	0.506486
2.0	128.0	128.0	64.0	0.364640	0.506923
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

Continued on next page

## B.1 Additional Figures and Tables

119

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
3.0	512.0	128.0	32.0	0.168366	0.525644
1.0	256.0	256.0	64.0	0.417803	0.525980
3.0	128.0	128.0	64.0	0.480340	0.526121
3.0	128.0	128.0	32.0	0.491876	0.527008
3.0	256.0	128.0	32.0	0.194120	0.528000
2.0	128.0	128.0	64.0	0.296537	0.528261
2.0	128.0	128.0	32.0	0.316390	0.529308
3.0	128.0	256.0	64.0	0.435649	0.529458
1.0	256.0	128.0	32.0	0.375717	0.529638
2.0	128.0	256.0	64.0	0.440450	0.529948
1.0	256.0	256.0	64.0	0.389651	0.531063
2.0	128.0	256.0	128.0	0.362561	0.533559
2.0	128.0	256.0	32.0	0.398919	0.533672
3.0	128.0	256.0	32.0	0.452009	0.536955
1.0	256.0	128.0	32.0	0.346140	0.538510
2.0	128.0	128.0	128.0	0.273516	0.539359
1.0	256.0	128.0	64.0	0.310597	0.539599
3.0	128.0	128.0	32.0	0.265842	0.539827
1.0	256.0	128.0	64.0	0.274568	0.541263
3.0	128.0	256.0	64.0	0.500697	0.544048
1.0	256.0	128.0	32.0	0.316189	0.545363
1.0	256.0	128.0	32.0	0.285714	0.546995
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

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
1.0	128.0	128.0	64.0	0.425469	0.555312
1.0	256.0	256.0	32.0	0.411686	0.555955
1.0	256.0	128.0	64.0	0.238860	0.556672
3.0	64.0	128.0	64.0	0.420250	0.559336
3.0	64.0	64.0	128.0	0.345705	0.559549
3.0	128.0	128.0	128.0	0.238071	0.562603
2.0	256.0	128.0	32.0	0.143647	0.563866
1.0	128.0	128.0	32.0	0.489160	0.564304
3.0	128.0	256.0	32.0	0.521478	0.566153
2.0	128.0	128.0	64.0	0.584950	0.567093
2.0	64.0	128.0	64.0	0.443393	0.567754
2.0	128.0	256.0	64.0	0.549169	0.568419
1.0	128.0	64.0	32.0	0.359041	0.569011
3.0	128.0	256.0	64.0	0.573862	0.570873
1.0	128.0	128.0	32.0	0.525982	0.571859
3.0	64.0	128.0	128.0	0.408074	0.572306
1.0	128.0	128.0	32.0	0.467434	0.572480
1.0	128.0	128.0	32.0	0.417764	0.573797
2.0	64.0	64.0	32.0	0.413944	0.573993
3.0	64.0	64.0	64.0	0.355615	0.574236
1.0	256.0	128.0	128.0	0.204964	0.574585
1.0	128.0	64.0	64.0	0.328927	0.575464
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

Continued on next page

## B.1 Additional Figures and Tables

## 121

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
1.0	128.0	256.0	64.0	0.501138	0.593823
3.0	64.0	32.0	128.0	0.339394	0.594401
1.0	128.0	32.0	32.0	0.348193	0.595001
2.0	64.0	128.0	32.0	0.470837	0.597005
3.0	64.0	32.0	64.0	0.344404	0.597406
2.0	128.0	64.0	64.0	0.224014	0.597418
1.0	64.0	32.0	64.0	0.462827	0.597437
1.0	64.0	32.0	32.0	0.500014	0.598521
2.0	64.0	32.0	128.0	0.376624	0.600570
1.0	64.0	32.0	128.0	0.453646	0.604043
1.0	128.0	256.0	64.0	0.539087	0.604710
2.0	256.0	128.0	64.0	0.122328	0.606237
1.0	64.0	128.0	128.0	0.489255	0.607122
1.0	128.0	32.0	64.0	0.319029	0.609441
1.0	128.0	256.0	64.0	0.566182	0.610409
1.0	128.0	32.0	128.0	0.294204	0.613838
1.0	64.0	64.0	128.0	0.436633	0.615036
1.0	64.0	64.0	64.0	0.461935	0.616265
2.0	128.0	64.0	128.0	0.206896	0.620845
1.0	128.0	256.0	32.0	0.550056	0.627652
2.0	256.0	128.0	128.0	0.106181	0.631364
3.0	128.0	64.0	32.0	0.185779	0.633145
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

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
1.0	64.0	128.0	32.0	0.561724	0.670482
1.0	256.0	64.0	128.0	0.149129	0.672432
2.0	128.0	32.0	64.0	0.193615	0.688310
2.0	128.0	128.0	64.0	0.802259	0.696580
2.0	128.0	256.0	32.0	0.907374	0.701893
3.0	256.0	128.0	128.0	0.076598	0.711632
2.0	256.0	64.0	32.0	0.081134	0.716840
2.0	128.0	32.0	128.0	0.173684	0.727354
2.0	256.0	64.0	64.0	0.073675	0.742250
1.0	256.0	32.0	32.0	0.161496	0.743529
3.0	128.0	32.0	32.0	0.146775	0.752404
1.0	256.0	32.0	64.0	0.138145	0.755407
1.0	256.0	32.0	128.0	0.125931	0.757801
3.0	128.0	32.0	64.0	0.134530	0.770094
2.0	256.0	64.0	128.0	0.063084	0.797383
3.0	128.0	32.0	128.0	0.129410	0.801131
3.0	256.0	64.0	64.0	0.048852	0.823713
3.0	256.0	64.0	32.0	0.052363	0.848516
2.0	256.0	32.0	32.0	0.058634	0.874037
3.0	256.0	64.0	128.0	0.044448	0.876398
2.0	256.0	32.0	128.0	0.049791	0.888397
2.0	256.0	32.0	64.0	0.050012	0.898488
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

## B.1 Additional Figures and Tables

123

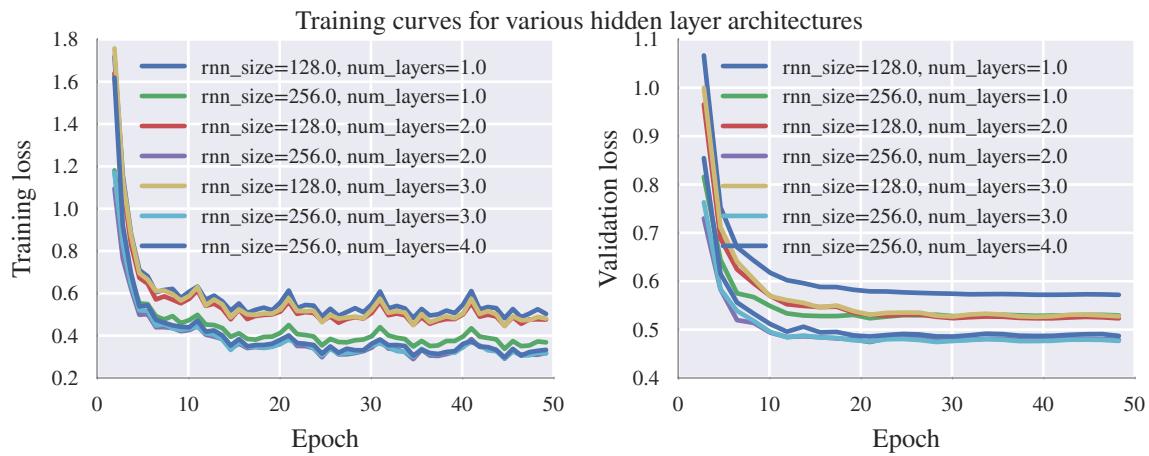


Fig. B.2 `rnn_size=256` and `num_layers=3` yields lowest validation loss.

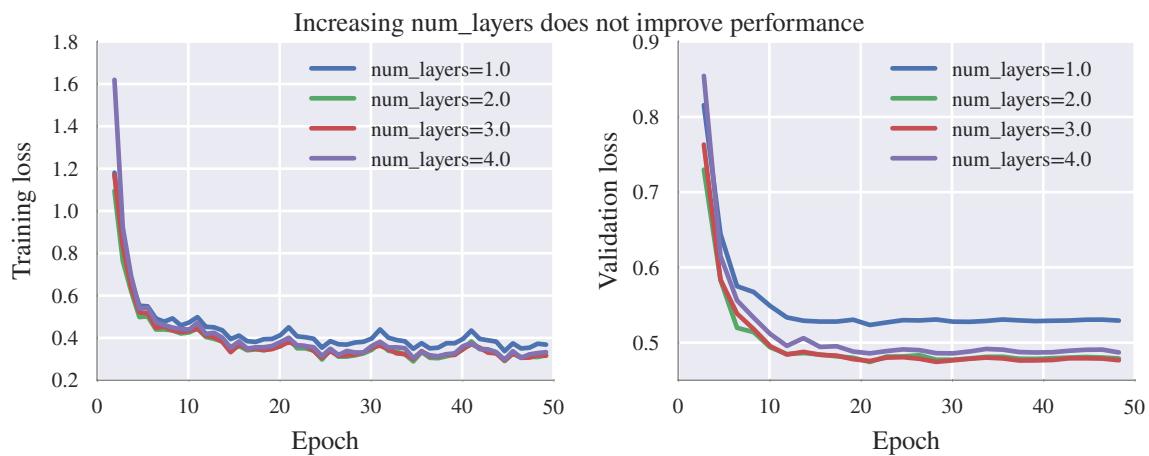


Fig. B.3 Validation loss improves initially with increasing network depth but deteriorates after > 3 layers.

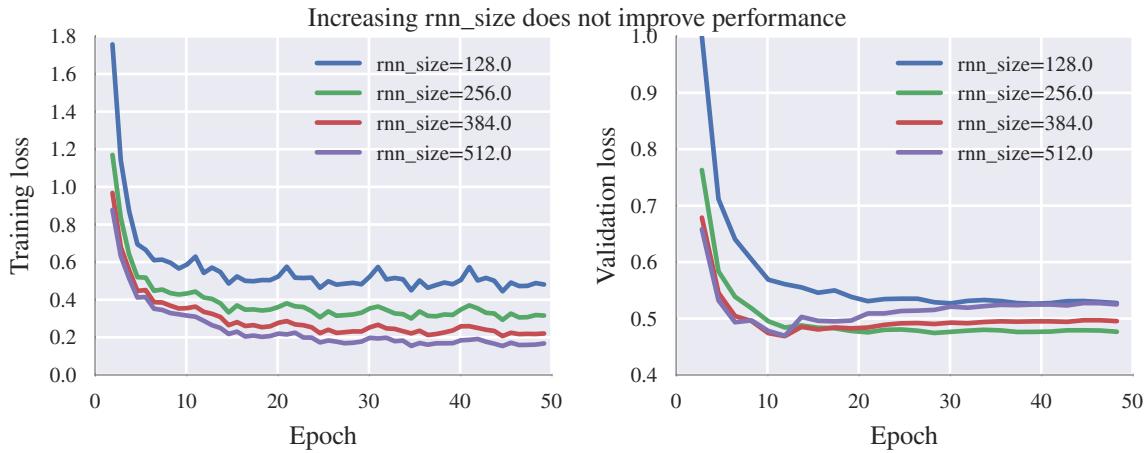


Fig. B.4 Validation loss improves initially with higher-dimensional hidden states but deteriorates after  $> 256$  dimensions.

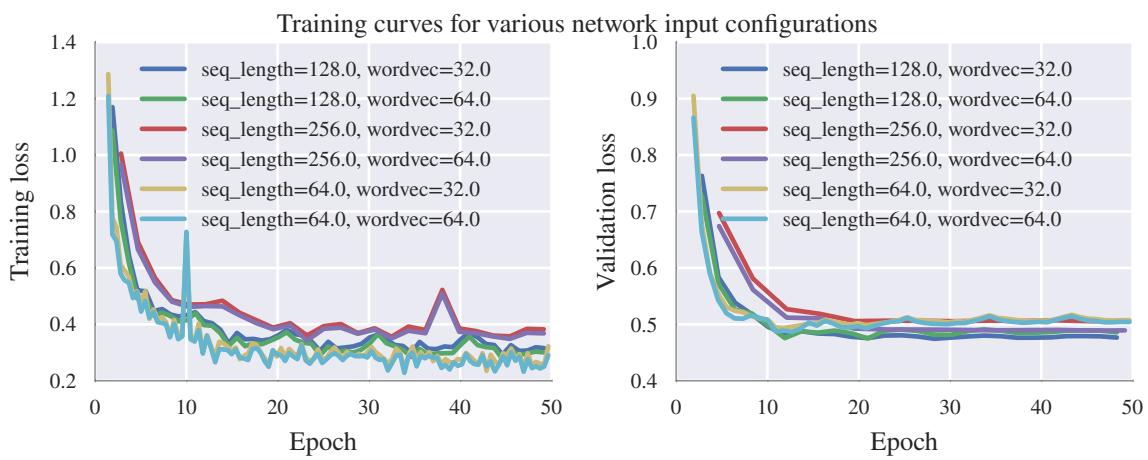


Fig. B.5  $\text{seq\_length}=128$  and  $\text{wordvec}=32$  yields lowest validation loss.

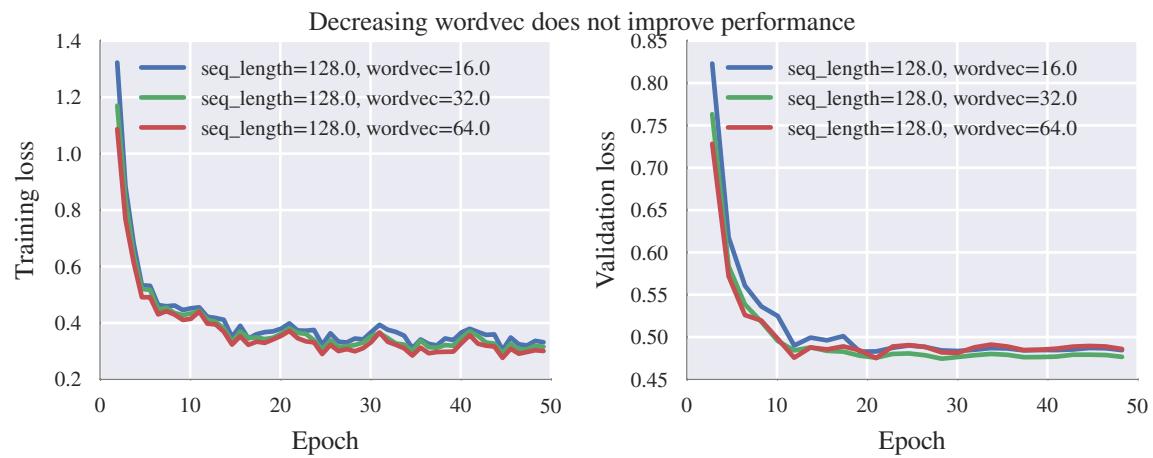
B.1 Additional Figures and Tables**125**

Fig. B.6 Perturbations about wordvec=32 do not yield significant improvements.

Draft - v1.1

Thursday 11<sup>th</sup> August, 2016 – 04:42

# 9

## Graveyard

### 9.1 Neural Networks

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

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

fliang: CITE: fig:nn-layer

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

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

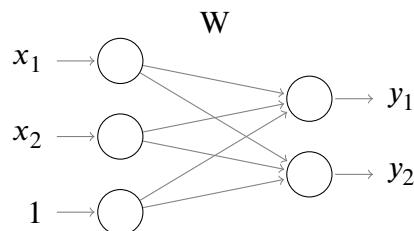


Fig. 9.1 Single feedforward neural network layer

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.*  $x^{(1)} = y^{(0)}$ ).

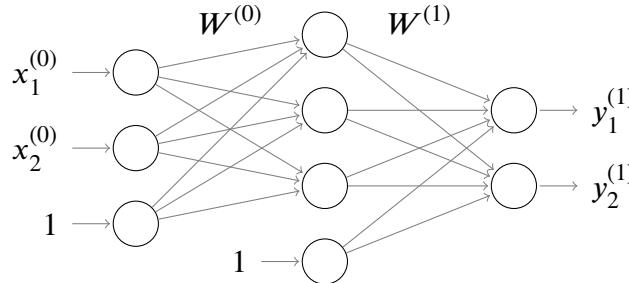


Fig. 9.2 2-layer feedforward neural network

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

The non-linearity introduced by the activation function  $\sigma$  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  $\mathbf{z}^{(L)}$  as the energies of a Boltzmann distribution (*i.e.* softmax). This implies a probability mass function on  $o$  given by

fliang: CITE: eq:softmax

.

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

## Efficient gradient computations through back-propagation

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

fliang: cite backprop

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

fliang: cite theano

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

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

fliang: Cite

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

## Vanishing gradients

The solution is to rewrite

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

such that

fliang: CITE: eq:prod-hi

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

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

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

## Training with back-propogation

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

fliang: CITE: eq:cross-entropy-loss

):

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

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

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

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

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

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

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

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

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

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

For earlier layers  $l < L$ , we have

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

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

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

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

fliang: DO THIS

$$\circ \quad (9.8)$$

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

The back-propagation algorithm consists of two steps:

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

2. **Backward pass:** Recursively iterate

fliang: CITE: eq:backprop

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

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

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

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

Some appealing properties of backpropogation:

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

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

## 9.2 RNNs

The advantages of RNNs over feedforward networks include:

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

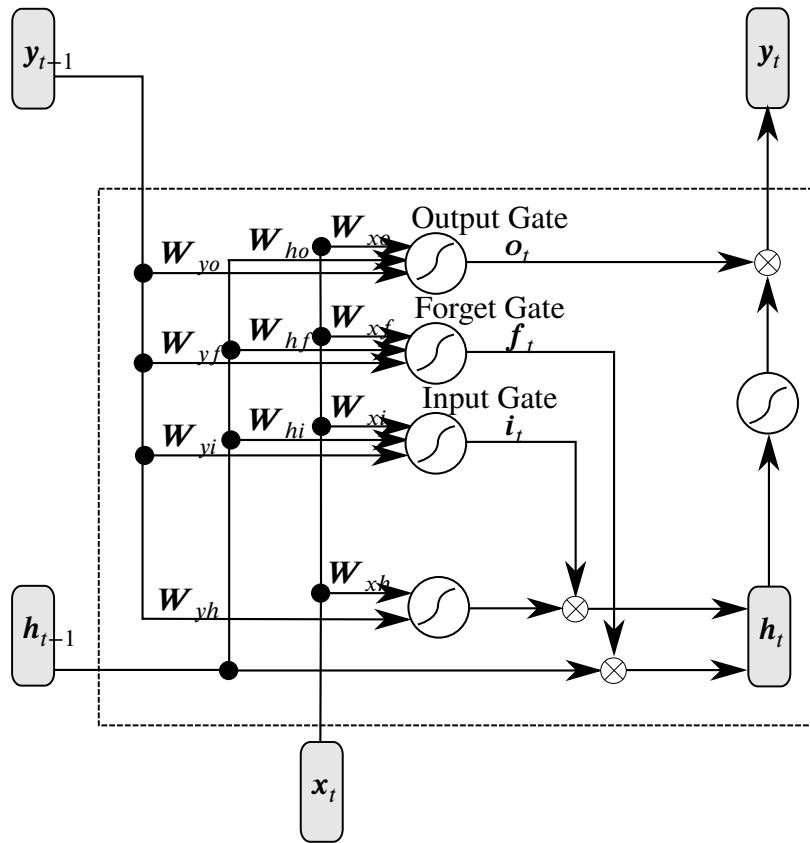


Fig. 9.3 Single LSTM unit

### Comparison against HMMs

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

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

## 9.3 Sequence probability modelling

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

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

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

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

## 6 **9.4 Related work**

7 [33] used music21 to generate rich feature representations for music for downstream machine  
8 learning tasks.

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

## 12 **9.5 LSTM: background and motivation**

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

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

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

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

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

27 [43] attempts to model meter by introducing time-delayed connections in [44]

### 28 **9.5.1 Representation of music data**

29 [96] discusses the importance of music representation, settling on **psychologically-based rep-**  
30 **resentations** of pitch, duration, and harmonic structure [118].

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

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

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

### 9.5.2 Evaluation of models

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

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

## 9.6 Automatic Composition

### 9.6.1 Multi-GPU implementation

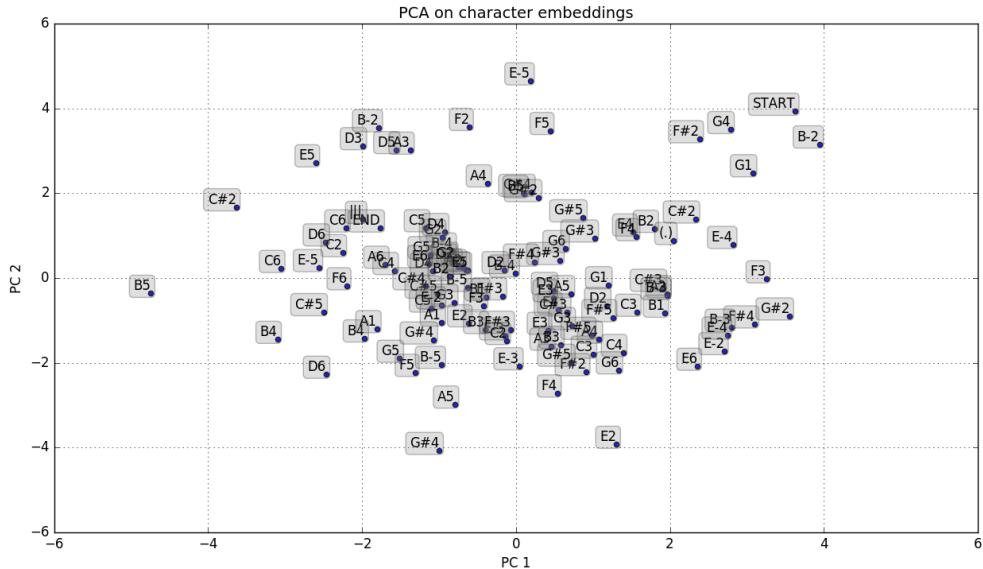
To accelerate model training, we parallelize models across multiple GPUs. This is possible thanks to the summation operation in noisy gradient estimators:

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

fliang: Real citations on noisy gradient

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

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



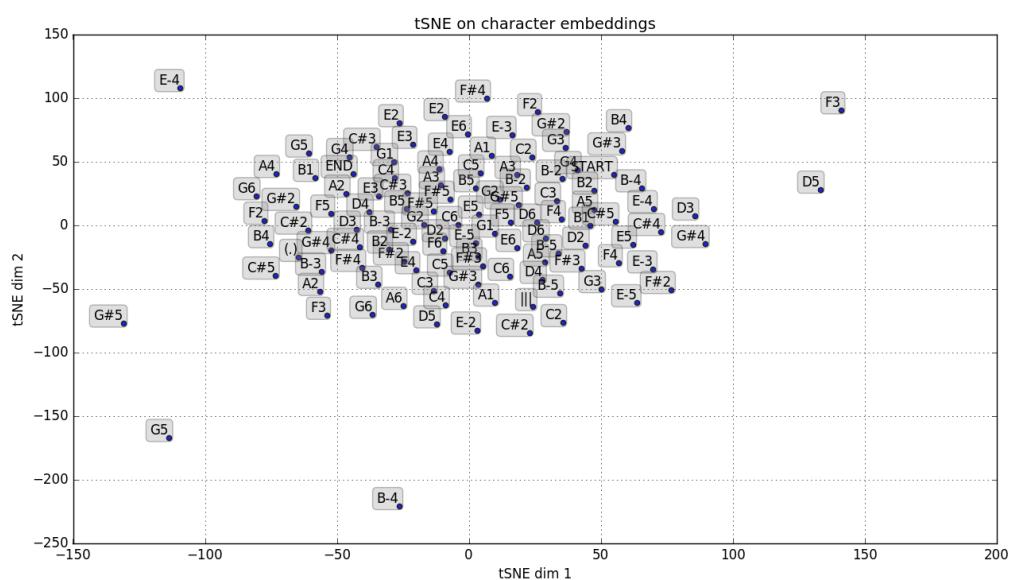
9.7 Token-level embeddings

Fig. 9.5 tSNE embedding of note tokens