3 Automatic stylistic composition with deep LSTM

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

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

Note that this is deviates from many prior works, which leveraged domain-specific knowledge such as modelling chords and notes hierarchically [20, 30, 13], accounting for meter [12], and detecting for motifs [14].

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.

3.1 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 [35] (*e.g.* no parallel fifths, voice leading) which can be use to qualitatively evaluate success

2. The large amount of easily recognizable structure: all chorales have exactly four parts consisting of a melody in the Soprano part harmonized by the Alto, Tenor, and Bass parts. Additionally, each chorale consists of a series of **phrases**: "groupings of consecutive notes into a unit that has complete musical sense of its own"'[31] which Bach delimited using fermatas

3. The Bach chorales have become a standardized corpus routinely studied by aspiring music theorists[40]

While the *JCB Chorales* [1] has become a popular dataset for polyphonic music modelling, we will show in 3.1.1 that its quantization to quavers introduces a non-negligible amount of distortortion.

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

Our data is obtained from the **Bach-Werke-Verzeichnis** (BWV) [9] indexed collection of the Bach chorales provided by the `music21`[11] Python library.

3.1.1 Preprocessing

Motivated by music's transposition invariance (see **??** on page ??) as well as prior practice [30, 13, 16, 15], we first perform **key normalization**. The keys of each score were first analyzed

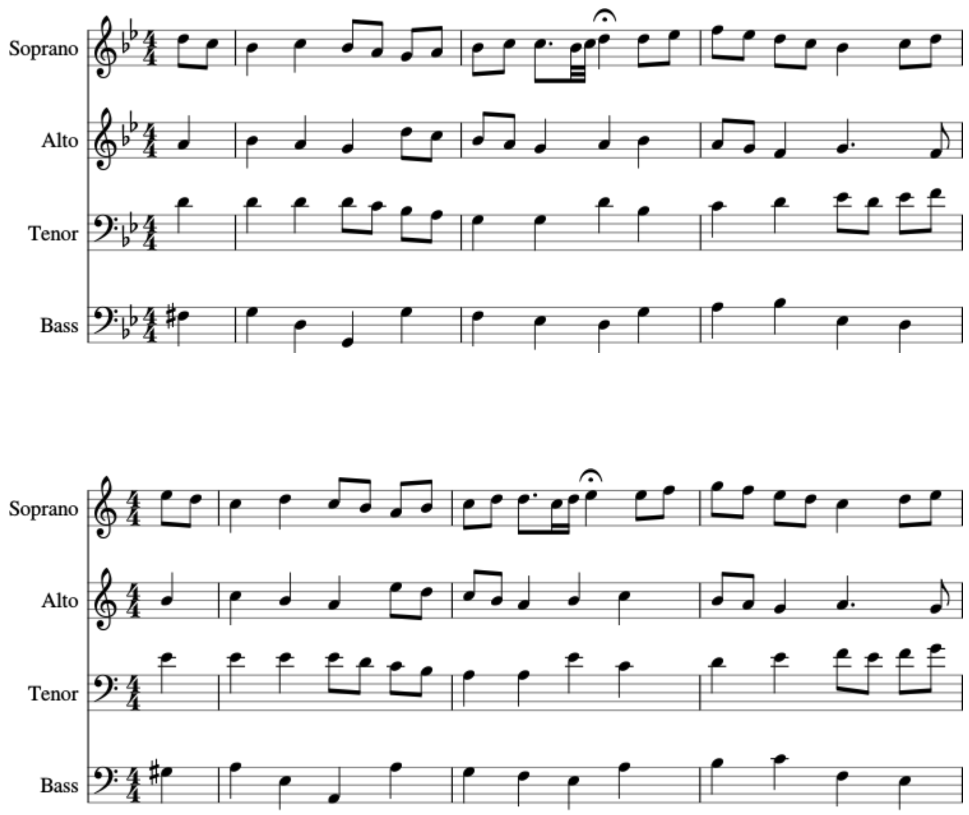


Fig. 3.1 First 4 bars of JCB Chorale BWV 185.6 before (top) and after (bottom) preprocessing. Note the transposition down by a semitone to C-major as well as quantization of the demisemiquavers in the third bar of the Soprano part.

using the Krumhansl Schmuckler key-finding algorithm [26] and then transposed such that the resulting score is C-major for major scores and A-minor for minor scores.

Next, **time quantization** is performed by aligning note start and end times to the nearest multiple of some fundamental duration. Our model uses a fundamental duration of one semibreve, **exceeding the time resolutions of [7, 13] by 2x, [20] by 4x, and [3] by 8x**.

We consider only note pitches and durations, neglecting changes in timing (*e.g.* ritardandos), dynamics (*e.g.* crescendos), and additional notation (*e.g.* accents, staccatos, legatos). This is comparable to prior work [7, 34] where a MIDI-encoding also lacking this additional notation was used.

An example of the effects introduced by our preprocessing is provided in fig. 3.1 in sheet music notation and in piano roll notation on fig. 3.2 on the following page.

fliang: Fix fig. 3.2 on the next page y-labels

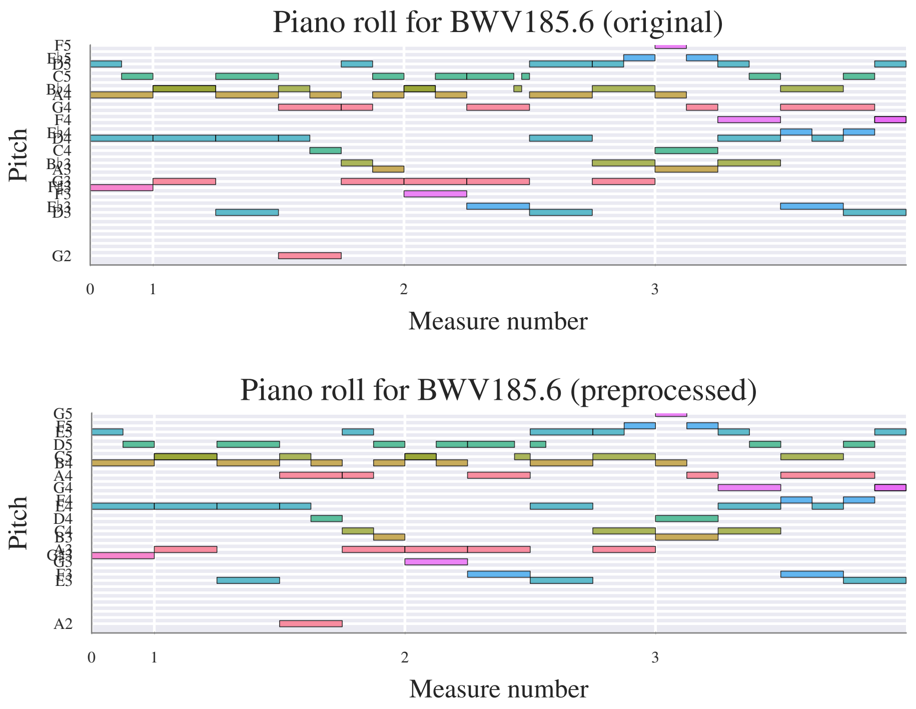


Fig. 3.2 Piano roll representation of the same 4 bars from fig. 3.1 before and after preprocessing. Again, note the transposition to C-major and time-quantization occuring in the Soprano part.

Quantizing to semiquavers introduces non-negligible distortion

Choosing to implement our own sequential encoding scheme was a difficult choice. While it would permit a finer time-resolution of semiquavers, it would make our cross-entropy losses incomparable to those reported on *JCB Chorales* [1].

To justify our decision, we investigated the distortion introduced by quantization to quavers rather than semiquavers in fig. 3.3 on the facing page. We find that *JCB Chorales* **distorts** 2816 **notes in the corpus** (2.85%) **because of quantization to quavers**. Since our research aim is to generate convincing music, we **minimize unnecessary distortions and proceed with our own encoding scheme**, understanding that it will create difficulties in comparing quantitative evaluation.

We also investigate changes in other corpus-level statistics as a result of key normalization and time quantization, such as pitch and pitch class usages and meter. All results fall within expectations, but the interested reader is directed to section A.1.1 on page 36.

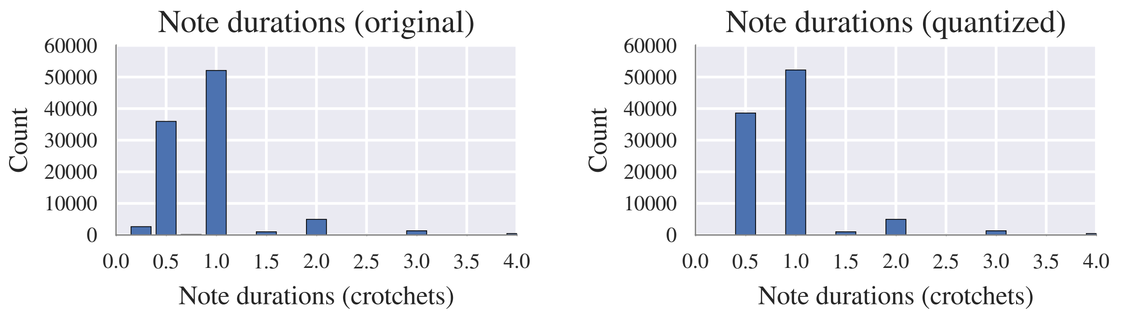


Fig. 3.3 Distortion introduced by quantization to semiquavers

3.1.2 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 [30, 16, 37]) or larger harmonic units (*e.g.* chords [13, 7], "harmonic context" [1]). This tradeoff is similar to one faced in RNN language modelling where either individual characters or entire words can be used.

In contrast to most language models which operate at the word level, we choose to construct our models at the note level. The use of **a note-level encoding may improve performance with respect to out-of-vocabulary (OOV) tokens** in two ways. It first reduces the potential vocabulary size from $O(128^4)$ possible chords down to $O(128)$ potential notes. In addition, harmonic relationships learned by the model parameters may enable generalization to OOV queries (*e.g.* OOV chords that are transpositions of in-vocabualry chords).

In fact, the decision may not even matter at all. Graves [18] showed comparable performance between LSTM language models that operate on individual characters versus words (perplexities of 1.24 bits vs 1.23 bits per character respectively), suggesting that choice of notes versus chords is not very significant, at least for English language modelling.

Similar to [39], our encoding represents polyphonic scores using a localist frame-based representation where time is discretized into constant timestep **frames**. Frame based processing forces the network to learn the relative duration of notes, a counting and timing task which [17] demonstrated LSTM is capable of. Consecutive frames are separated by a unique delimiter ("|||" in fig. 3.4 on the following page). Each frame consists of a sequence of $\langle$Note, Tie$\rangle$ tuples where Note $\in \{0, 1, \cdots, 127\}$ represents the MIDI pitch of a note and Tie $\in \{True, False\}$ distinguishes whether a note is tied with a note at the same pitch from the previous frame or is articulated at the current timestep.

A design decision is the order in which notes within a frame are encoded and consequentially processed by a sequential model. Since chorale music places the melody in the Soprano

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

Fig. 3.4 Example encoding of two chords, the second one annotated with fermatas (tokens 122 to 148, BWV 101.7). chords are encoded as (MIDI pitch value, tied to previous frame?) tuples, "|||"" encodes the ends of frames, and "(.)" at the start of a chord encodes a fermata. Each "|||" corresponds to time advancing by a semiquaver

Table 3.1 Statistics on the preprocessed datasets used throughout our study

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

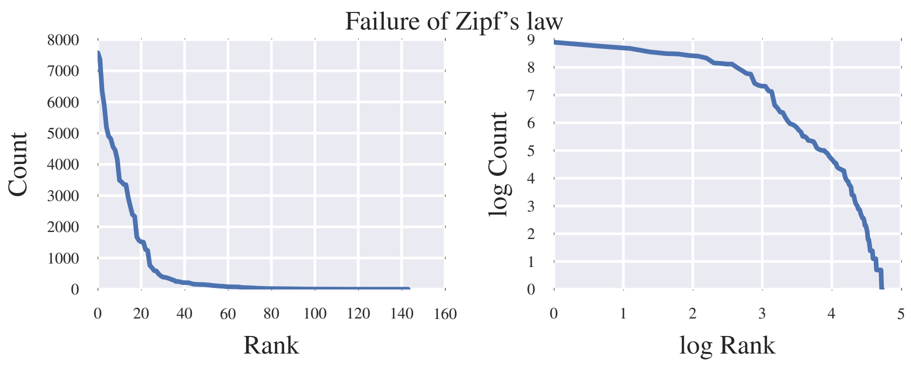


Fig. 3.5 Left: Token frequencies sorted by rank. Right: log-log plot where a power law distribution as predicted by Zipf's law would appear linear.

part, it is reasonable to expect the Soprano notes to be most significant in determining the other parts. Hence, we would like to process Soprano notes first and **order the notes within a frame in descending pitch**.

The above specification describes our initial attempt at an encoding format. However, we found that this encoding format resulted in unrealistically long phrase lengths. Including fermatas (represented by "(.)" in fig. 3.4 on page 22), which Bach used to denote ends of phrases, solves this problem.

Finally, for each score a unique start symbol ("START" in fig. 3.4) and end symbol ("END" in fig. 3.4) are appended to the beginning and end respectively. This causes the model to learn to initialize itself when given the start symbol and allows us to determine when a composition generated by the model has concluded.

The vocabulary and corpus size after encoding is detailed in table 3.1. The rank-size distribution of the note-level corpus tokens is shown in fig. 3.5 and confirms the failure of Zipf's law in our data.

We make the following observations about our proposed encoding scheme:

- It is **sparse**: unarticulated notes are not encoded
- It is also **variable length**: each frame can span anywhere from one to five tokens, requiring LSTM's capability of detecting spacing between events[17]

- The explicit representation of tied notes vs articulated notes **enables us to determine when notes end**, resolving an issue present in many prior works [13, 12, 28, 8]

Unlike many others [30, 16, 27], we avoid adding prior informaton through engineering harmonically relevant features. Instead, we appeal to results by Bengio [5] and Bengio and Delalleau [6] suggesting that that a key ingredient in deep learning's success is its **ability to learn good features from raw data**. Such features are very likely to be musically relevant, which we will explore further in **??**.

3.2 Design and validation of a generative model for music

In this section, we describe the design and validation process leading to our generative model.

3.2.1 Training and evaluation criteria

Following [30], we will train the model to predict $P(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$: a probability distribution over all possible next tokens $\boldsymbol{x}_{t+1}$ given the current token $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$. This is the exact same operation performed by RNN language models [29]. We minimize cross-entropy loss between the predicted distributions $P(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$ and the actual target distribution $\delta_{\boldsymbol{x}_{t+1}}$

At the next timestep, the correct token $\boldsymbol{x}_{t+1}$ is provided as the recurrent input even if the most likely prediction argmax $P(\boldsymbol{x}_{t+1}|\boldsymbol{h}_t, \boldsymbol{x}_t)$ differs. This is is referred to as **teacher forcing** [41] performed to aid convergence because the model's predictions may not be reliable early in training.

However, at inference the token generated from $P(\boldsymbol{x}_{t+1}|\boldsymbol{h}_t, \boldsymbol{x}_t)$ is reused as the previous input, creating a discrepancy between training and inference. Scheduled sampling [4] is a recently proposed alternative training method for resolving this discrepancy and may help the model better learn to predict using generated symbols rather than relying on ground truth to be always provided as input.

3.2.2 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 [19] and SRILM [36], as baselines. KenLM implements an efficient modified Kneser-Ney smoothing language model and while SRILM provides a variety of language models we choose choose to use the Good-Turing discounted language model for benchmarking against.

Both models were developed for applications modelling language data, whose distribution over words which may differ from our encoded music data (see fig. 3.5 on page 23). Furthermore, both are based upon N-gram models which are constrained to only account for short-term dependencies. Therefore, we expect RNNs to outperform the N-gram baselines shown in table 3.2 on the next page.

3.2.3 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` – number of frames before truncating BPTT gradient
5. `dropout` – the dropout probability

fliang: Does this need to be diagrammed?

Our model first embeds the inputs $\boldsymbol{x}_t$ into a `wordvec`-dimensional vector-space, compressing the dimensionality down from $|V| \approx 140$ to `wordvec` dimensions. Next, `num_layers` layers of memory cells followed by batch normalization [23] and dropout [21] with dropout probability `dropout` are stacked. The outputs $\boldsymbol{y}_t^{(\texttt{num\_layers})}$ are followed by a fully-connected layer mapping to $|V| = 108$ units, which are passed through a softmax to yield a predictive distribution $P(\boldsymbol{x}_{t+1}|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$. Cross entropy is used as the loss minimized during training.

Models were trained using Adam [25] with an initial learning rate of 2×10^{-3} decayed by 0.5 every 5 epochs. The back-propogation through time gradients were clipped at ± 5.0 [33] and BPTT was truncated after `seq_length` frames. A minibatch size of 50 was used.

3.2.4 Comparison of memory cells on music data

We used `theanets`[1] to rapidly implement and compare a large number of memory cell implementations. Figure 3.6 shows the results of exploring a range of RNN memory cell implementation and holding `num_layers=1`, `rnn_size=130`, `wordvec=64`, and `seq_length=50`

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

Table 3.2 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

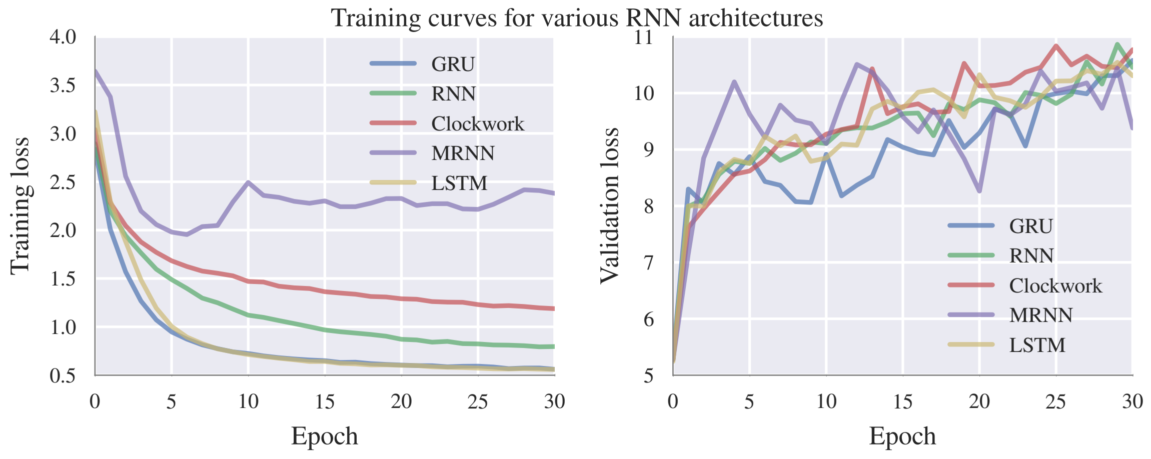


Fig. 3.6 LSTM and GRUs yield the lowest training loss. Validation loss traces show all architectures exhibit signs of significant overfitting

constant. Unlike later models, none of these models utilized dropout or batch normalization. We configured the clockwork RNN [10] with 5 equal-sized hidden state blocks with update periods $(1, 2, 4, 8, 16)$.

Figure 3.6 shows that while all models achieved similar validation losses, LSTM and GRUs trained much faster and achieved lower training loss. Since Zaremba [42] find similar empirical performance between LSTM and GRUs and Nayebi and Vitelli [32] 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. 3.6 is a red flag suggesting that overfitting is occuring. This observation motivates the exporation of dropout regularization in section 3.2.5.

3.2.5 Optimizing the LSTM architecture

After settling on LSTM as the memory cell, we conducted remaining experiments using the `torch-rnn` Lua software library. Our switch was motivated by support for GPU training (see table 3.3 on page 30), dropout, and batch normalization.

Dropout regularization improves validation loss

The increasing validation errors in fig. 3.6 prompted investigation of regularization techniques. In addition to adding batch normalization, a technique known to reduce overfitting and accelerate training [23], we also investigated the effects of different levels of dropout by varying the `dropout` parameter.

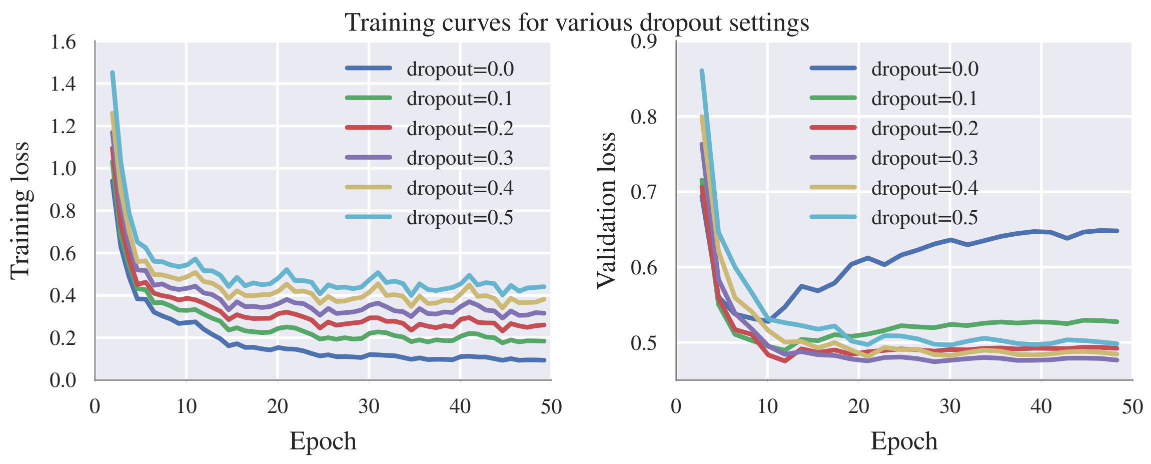


Fig. 3.7 Dropout acts as a regularizer, resulting in larger training loss but better generalization as evidenced by lower validation loss. A setting of `dropout=0.3` achieves best results for our model.

The experimental results are shown in fig. 3.7. As expected, dropout acts as a regularizer and reduces validation loss from 0.65 down to 0.477 (when `dropout=0.3`). Training loss has slightly increased, which is also unexpected as application of dropout during training introduces additional noise into the model.

Overall best model

We perform a grid search through the following parameter grid:

- `num_layers` $\in \{1, 2, 3, 4\}$
- `rnn_size` $\in \{128, 256, 384, 512\}$
- `wordvec` $\in \{16, 32, 64\}$
- `seq_length` $\in \{64, 128, 256\}$
- `dropout` $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$

A full listing of results is provided in fig. A.5 on page 38.

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

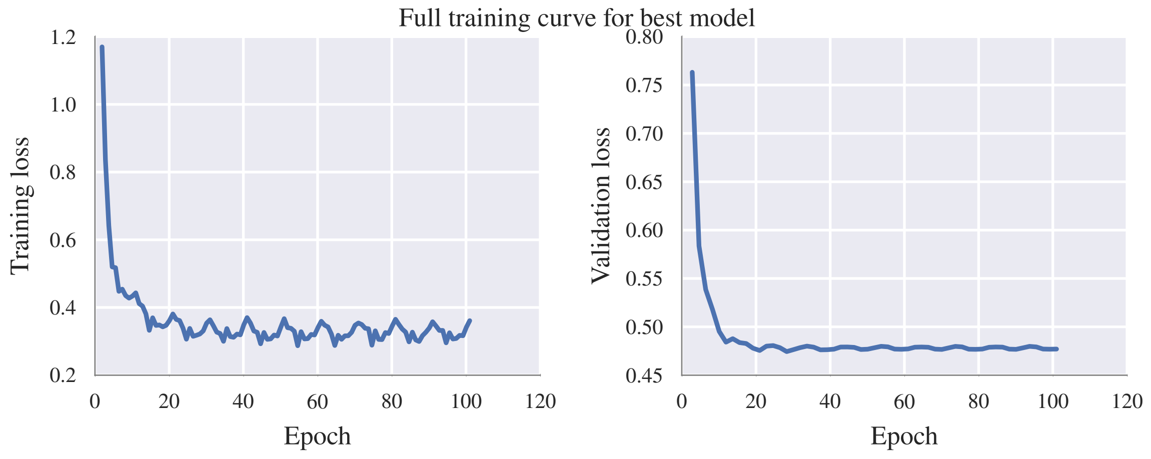


Fig. 3.8 Training curves for the overall best model. The periodic spikes correspond to resetting of the LSTM state at the end of a training epoch.

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

1. Depth matters! Increasing `num_layers` can yield up to 9% lower validation loss. The best model is 3 layers deep, any further and overfitting occurs. This finding is unsurprising: the dominance of deep RNNs in polyphonic modelling was already noted by Pascanu et al. [34]

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

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

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

3.2.6 GPU training yields 800% acceperation

Consistent with prior work [38, 24], timing results table 3.3 from training our overall best model confirmed a 800% speedup enabled by the GPU training implemented in `torch-rnn`.

3.3 Results and comparison

As done by [2, 7], we quantitatively evaluate our models using cross entropies and perplexities on a 10% held-out validation set. Our best model (fig. A.5 on page 38) achieves **cross-entropy**

Table 3.3 Timing results comparing CPU and GPU training of the overall best model (section 3.2.5 on page 28)

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

losses of 0.323 **on training data and** 0.477 **on held-out test data, corresponding to a training perplexity of** 1.251 **bits and a test perplexity of** 1.391. As expected, the deep LSTM model achieves more than 0.6 **bits lower than any validation perplexity obtained by the** N**-gram models** compared in table 3.2 on page 26.

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 [22]

References

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

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

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

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

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

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

[7] 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.

[8] 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.

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

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

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

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

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

[14] 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.

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

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

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

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

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

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

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

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

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

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

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

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

[27] 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.

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

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

[30] 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.

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

[32] 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.

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

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

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

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

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

[38] 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.

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

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

[41] 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.

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

A

Appendix C: Additional Proofs, Figures, and Tables

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

A.1 Sufficient conditions for vanishing gradients

Following Pascanu, Mikolov, and Bengio [33], let $\|\cdot\|$ be any submultiplicative matrix norm (*e.g.* Frobenius, spectral, nuclear, Shatten p-norms). Without loss of generality, we will use the **operator norm** defined as

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

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

Applying the definition of submultiplicativity to the factors of the product in eq. (1.4), we have that for any k

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

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

$$\begin{aligned}
\gamma_\sigma &\coloneqq \sup_{h\in\mathbb{R}^n} \|\operatorname{diag}\left(\sigma'_{hh}(\boldsymbol{h})\right)\| && && \text{(A.3)}\\
&= \sup_{h\in\mathbb{R}^n} \max_i \sigma'_{hh}(\boldsymbol{h})_i && \text{Operator norm of diag} && \text{(A.4)}\\
&= \sup_{x\in\mathbb{R}} \sigma'_{hh}(x) && \sigma_{hh} \text{ acts elementwise} && \text{(A.5)}
\end{aligned}$$

Substituting back into eq. (1.4), we find that

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

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

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

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

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

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

A.1.1 Quantifying the effects of preprocessing

Related discussion is in section 3.1.1 on page 18.

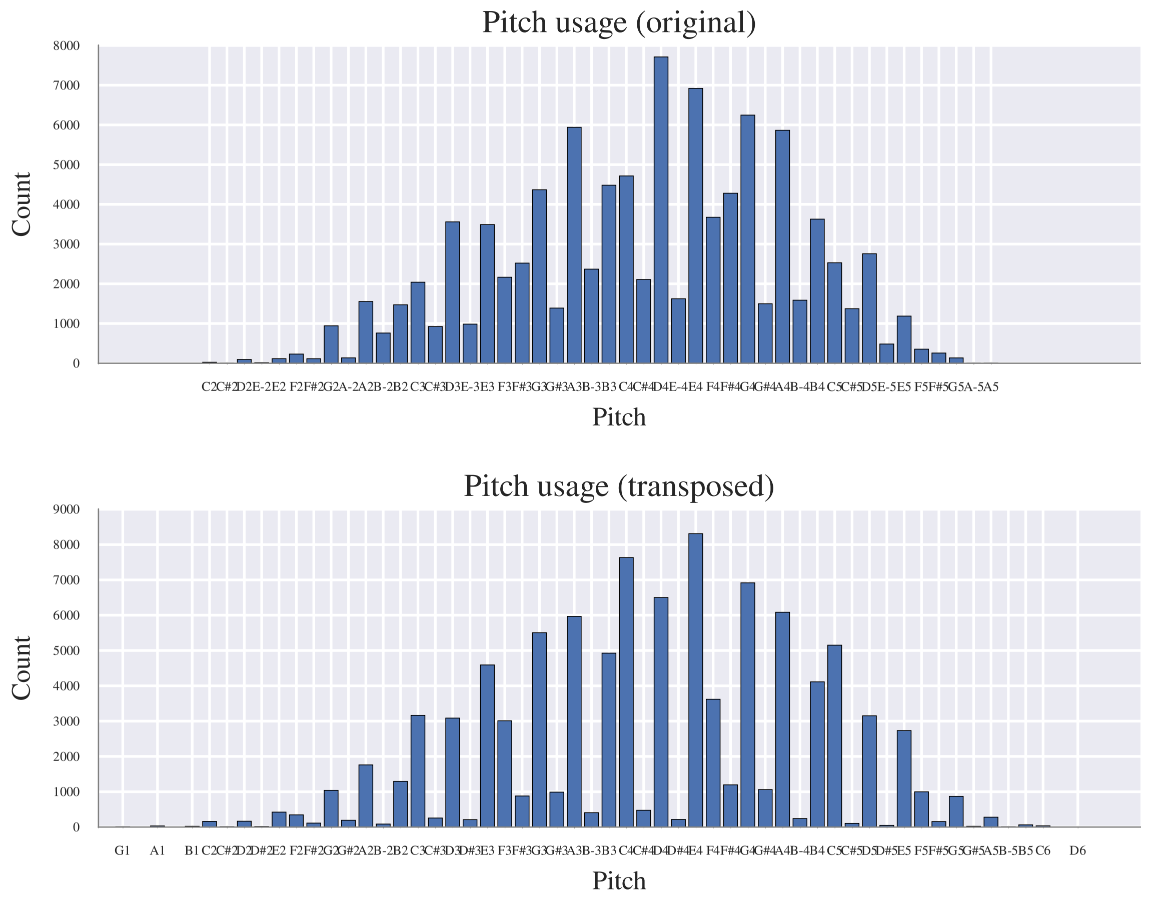


Fig. A.1 Distribution of pitches used over Bach chorales corpus. Transposition has resulted in an overall broader range of pitches and increased the counts of pitches which are in key.

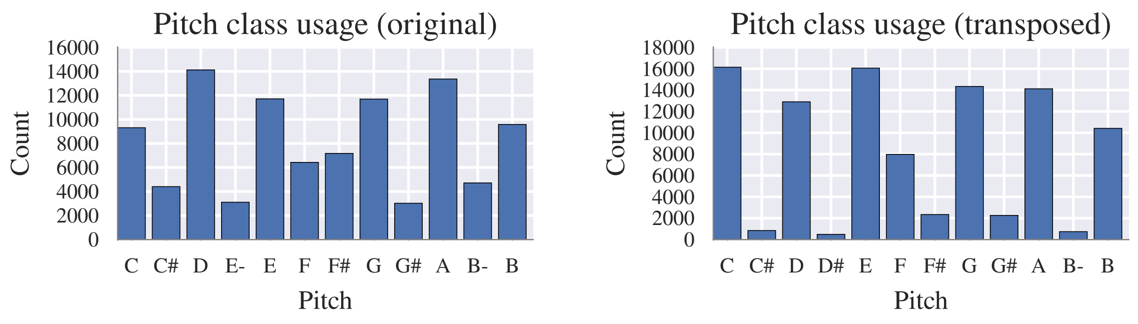


Fig. A.2 Distribution of pitch classes over Bach chorales corpus. Transposition has increased the counts for pitch classes within the C-major / A-minor scales.

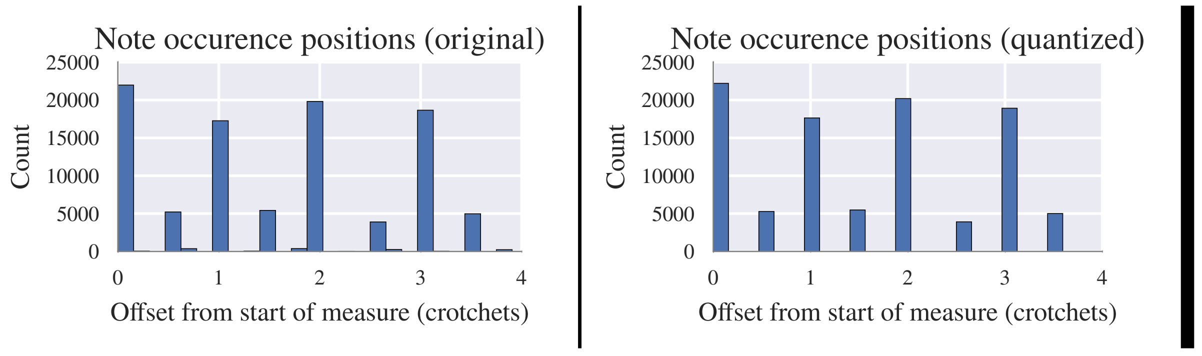


Fig. A.3 Meter is minimally affected by quantization due to the high resolution used for time quantization.

A.1.2 Discovering neurons specific to musical concepts

Related discussion is in **??** on page ??.

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

Related discussion is in section 3.2.5 on page 28.

Fig. A.5 Results of grid search (see Section 3.2.5) 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

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
2.0	256.0	128.0	32.0	0.323668	0.479322
2.0	256.0	128.0	64.0	0.303158	0.482216
3.0	256.0	256.0	64.0	0.320361	0.484231
3.0	256.0	128.0	32.0	0.383811	0.484667
3.0	256.0	128.0	16.0	0.342955	0.484791
2.0	256.0	256.0	64.0	0.373641	0.485353
3.0	256.0	128.0	64.0	0.305290	0.486244
2.0	256.0	128.0	32.0	0.275125	0.486305
2.0	256.0	256.0	32.0	0.352257	0.486755
4.0	256.0	128.0	32.0	0.333133	0.487135
2.0	256.0	256.0	32.0	0.307188	0.487868
2.0	256.0	256.0	32.0	0.400955	0.489320
3.0	256.0	256.0	64.0	0.381868	0.489810
2.0	256.0	256.0	64.0	0.333356	0.491396
2.0	256.0	256.0	64.0	0.284248	0.491593
3.0	128.0	128.0	32.0	0.365171	0.492478
3.0	256.0	128.0	32.0	0.264723	0.492849
3.0	384.0	128.0	32.0	0.228556	0.495991
3.0	256.0	128.0	64.0	0.248987	0.496190
3.0	256.0	128.0	32.0	0.445840	0.498205
3.0	256.0	256.0	32.0	0.273567	0.499422
2.0	256.0	128.0	64.0	0.256022	0.500500
3.0	256.0	256.0	32.0	0.338776	0.501711
2.0	128.0	128.0	32.0	0.384075	0.501840
3.0	128.0	128.0	64.0	0.417780	0.501919
2.0	256.0	128.0	32.0	0.219939	0.502503
3.0	128.0	128.0	64.0	0.361381	0.503206
3.0	128.0	128.0	32.0	0.431771	0.503590
3.0	256.0	64.0	64.0	0.263001	0.503945
3.0	256.0	384.0	64.0	0.419091	0.504249
3.0	256.0	256.0	32.0	0.393463	0.506486
2.0	128.0	128.0	64.0	0.364640	0.506923
2.0	128.0	128.0	64.0	0.422178	0.507268

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
3.0	256.0	256.0	64.0	0.261563	0.507479
3.0	256.0	64.0	32.0	0.278916	0.507673
2.0	128.0	128.0	32.0	0.434552	0.508460
3.0	256.0	384.0	32.0	0.439684	0.514804
1.0	256.0	128.0	64.0	0.334873	0.517134
2.0	128.0	128.0	64.0	0.465061	0.520224
2.0	256.0	128.0	64.0	0.195905	0.521330
1.0	256.0	256.0	64.0	0.368281	0.522424
2.0	128.0	128.0	32.0	0.485346	0.522955
2.0	128.0	256.0	64.0	0.378280	0.525397
3.0	512.0	128.0	32.0	0.168366	0.525644
1.0	256.0	256.0	64.0	0.417803	0.525980
3.0	128.0	128.0	64.0	0.480340	0.526121
3.0	128.0	128.0	32.0	0.491876	0.527008
3.0	256.0	128.0	32.0	0.194120	0.528000
2.0	128.0	128.0	64.0	0.296537	0.528261
2.0	128.0	128.0	32.0	0.316390	0.529308
3.0	128.0	256.0	64.0	0.435649	0.529458
1.0	256.0	128.0	32.0	0.375717	0.529638
2.0	128.0	256.0	64.0	0.440450	0.529948
1.0	256.0	256.0	64.0	0.389651	0.531063
2.0	128.0	256.0	128.0	0.362561	0.533559
2.0	128.0	256.0	32.0	0.398919	0.533672
3.0	128.0	256.0	32.0	0.452009	0.536955
1.0	256.0	128.0	32.0	0.346140	0.538510
2.0	128.0	128.0	128.0	0.273516	0.539359
1.0	256.0	128.0	64.0	0.310597	0.539599
3.0	128.0	128.0	32.0	0.265842	0.539827
1.0	256.0	128.0	64.0	0.274568	0.541263
3.0	128.0	256.0	64.0	0.500697	0.544048
1.0	256.0	128.0	32.0	0.316189	0.545363
1.0	256.0	128.0	32.0	0.285714	0.546995
3.0	128.0	128.0	64.0	0.247192	0.549826

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
1.0	128.0	128.0	64.0	0.458142	0.550102
1.0	128.0	128.0	128.0	0.360038	0.550509
2.0	128.0	256.0	32.0	0.465110	0.550995
1.0	256.0	256.0	32.0	0.444180	0.551894
3.0	256.0	128.0	64.0	0.184959	0.552200
2.0	128.0	256.0	64.0	0.490587	0.552217
2.0	128.0	256.0	32.0	0.514900	0.553092
1.0	128.0	128.0	64.0	0.487574	0.553498
1.0	256.0	256.0	32.0	0.471938	0.553586
1.0	128.0	128.0	64.0	0.384282	0.554990
1.0	128.0	128.0	64.0	0.425469	0.555312
1.0	256.0	256.0	32.0	0.411686	0.555955
1.0	256.0	128.0	64.0	0.238860	0.556672
3.0	64.0	128.0	64.0	0.420250	0.559336
3.0	64.0	64.0	128.0	0.345705	0.559549
3.0	128.0	128.0	128.0	0.238071	0.562603
2.0	256.0	128.0	32.0	0.143647	0.563866
1.0	128.0	128.0	32.0	0.489160	0.564304
3.0	128.0	256.0	32.0	0.521478	0.566153
2.0	128.0	128.0	64.0	0.584950	0.567093
2.0	64.0	128.0	64.0	0.443393	0.567754
2.0	128.0	256.0	64.0	0.549169	0.568419
1.0	128.0	64.0	32.0	0.359041	0.569011
3.0	128.0	256.0	64.0	0.573862	0.570873
1.0	128.0	128.0	32.0	0.525982	0.571859
3.0	64.0	128.0	128.0	0.408074	0.572306
1.0	128.0	128.0	32.0	0.467434	0.572480
1.0	128.0	128.0	32.0	0.417764	0.573797
2.0	64.0	64.0	32.0	0.413944	0.573993
3.0	64.0	64.0	64.0	0.355615	0.574236
1.0	256.0	128.0	128.0	0.204964	0.574585
1.0	128.0	64.0	64.0	0.328927	0.575464
2.0	64.0	64.0	64.0	0.390597	0.575592

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
2.0	64.0	128.0	128.0	0.424735	0.575868
2.0	64.0	32.0	32.0	0.399389	0.577974
2.0	64.0	64.0	128.0	0.372478	0.578856
2.0	128.0	64.0	32.0	0.240288	0.580802
3.0	64.0	64.0	32.0	0.375478	0.582072
1.0	128.0	64.0	128.0	0.304245	0.582897
3.0	64.0	128.0	32.0	0.430421	0.582991
3.0	128.0	256.0	32.0	0.590133	0.585245
3.0	64.0	32.0	32.0	0.348150	0.585800
2.0	64.0	32.0	64.0	0.387047	0.589173
1.0	128.0	256.0	64.0	0.501138	0.593823
3.0	64.0	32.0	128.0	0.339394	0.594401
1.0	128.0	32.0	32.0	0.348193	0.595001
2.0	64.0	128.0	32.0	0.470837	0.597005
3.0	64.0	32.0	64.0	0.344404	0.597406
2.0	128.0	64.0	64.0	0.224014	0.597418
1.0	64.0	32.0	64.0	0.462827	0.597437
1.0	64.0	32.0	32.0	0.500014	0.598521
2.0	64.0	32.0	128.0	0.376624	0.600570
1.0	64.0	32.0	128.0	0.453646	0.604043
1.0	128.0	256.0	64.0	0.539087	0.604710
2.0	256.0	128.0	64.0	0.122328	0.606237
1.0	64.0	128.0	128.0	0.489255	0.607122
1.0	128.0	32.0	64.0	0.319029	0.609441
1.0	128.0	256.0	64.0	0.566182	0.610409
1.0	128.0	32.0	128.0	0.294204	0.613838
1.0	64.0	64.0	128.0	0.436633	0.615036
1.0	64.0	64.0	64.0	0.461935	0.616265
2.0	128.0	64.0	128.0	0.206896	0.620845
1.0	128.0	256.0	32.0	0.550056	0.627652
2.0	256.0	128.0	128.0	0.106181	0.631364
3.0	128.0	64.0	32.0	0.185779	0.633145
1.0	128.0	256.0	32.0	0.591930	0.638022

Continued on next page

num_layers	rnn_size	seq_length	wordvec	train_metric	val_metric
1.0	256.0	64.0	32.0	0.200897	0.640652
1.0	64.0	64.0	32.0	0.487779	0.643943
1.0	128.0	256.0	32.0	0.621720	0.647467
2.0	128.0	32.0	32.0	0.209044	0.647553
3.0	256.0	128.0	32.0	0.100153	0.650138
1.0	64.0	128.0	64.0	0.515733	0.653191
1.0	256.0	64.0	64.0	0.171567	0.657626
3.0	256.0	128.0	64.0	0.087426	0.660995
3.0	128.0	64.0	128.0	0.169560	0.663409
3.0	128.0	64.0	64.0	0.172871	0.670402
1.0	64.0	128.0	32.0	0.561724	0.670482
1.0	256.0	64.0	128.0	0.149129	0.672432
2.0	128.0	32.0	64.0	0.193615	0.688310
2.0	128.0	128.0	64.0	0.802259	0.696580
2.0	128.0	256.0	32.0	0.907374	0.701893
3.0	256.0	128.0	128.0	0.076598	0.711632
2.0	256.0	64.0	32.0	0.081134	0.716840
2.0	128.0	32.0	128.0	0.173684	0.727354
2.0	256.0	64.0	64.0	0.073675	0.742250
1.0	256.0	32.0	32.0	0.161496	0.743529
3.0	128.0	32.0	32.0	0.146775	0.752404
1.0	256.0	32.0	64.0	0.138145	0.755407
1.0	256.0	32.0	128.0	0.125931	0.757801
3.0	128.0	32.0	64.0	0.134530	0.770094
2.0	256.0	64.0	128.0	0.063084	0.797383
3.0	128.0	32.0	128.0	0.129410	0.801131
3.0	256.0	64.0	64.0	0.048852	0.823713
3.0	256.0	64.0	32.0	0.052363	0.848516
2.0	256.0	32.0	32.0	0.058634	0.874037
3.0	256.0	64.0	128.0	0.044448	0.876398
2.0	256.0	32.0	128.0	0.049791	0.888397
2.0	256.0	32.0	64.0	0.050012	0.898488
3.0	256.0	32.0	32.0	0.037417	0.960396

Continued on next page

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

A.1.4 Additional large-scale subjective evaluation results

Related discussion is in **??** on page ??.

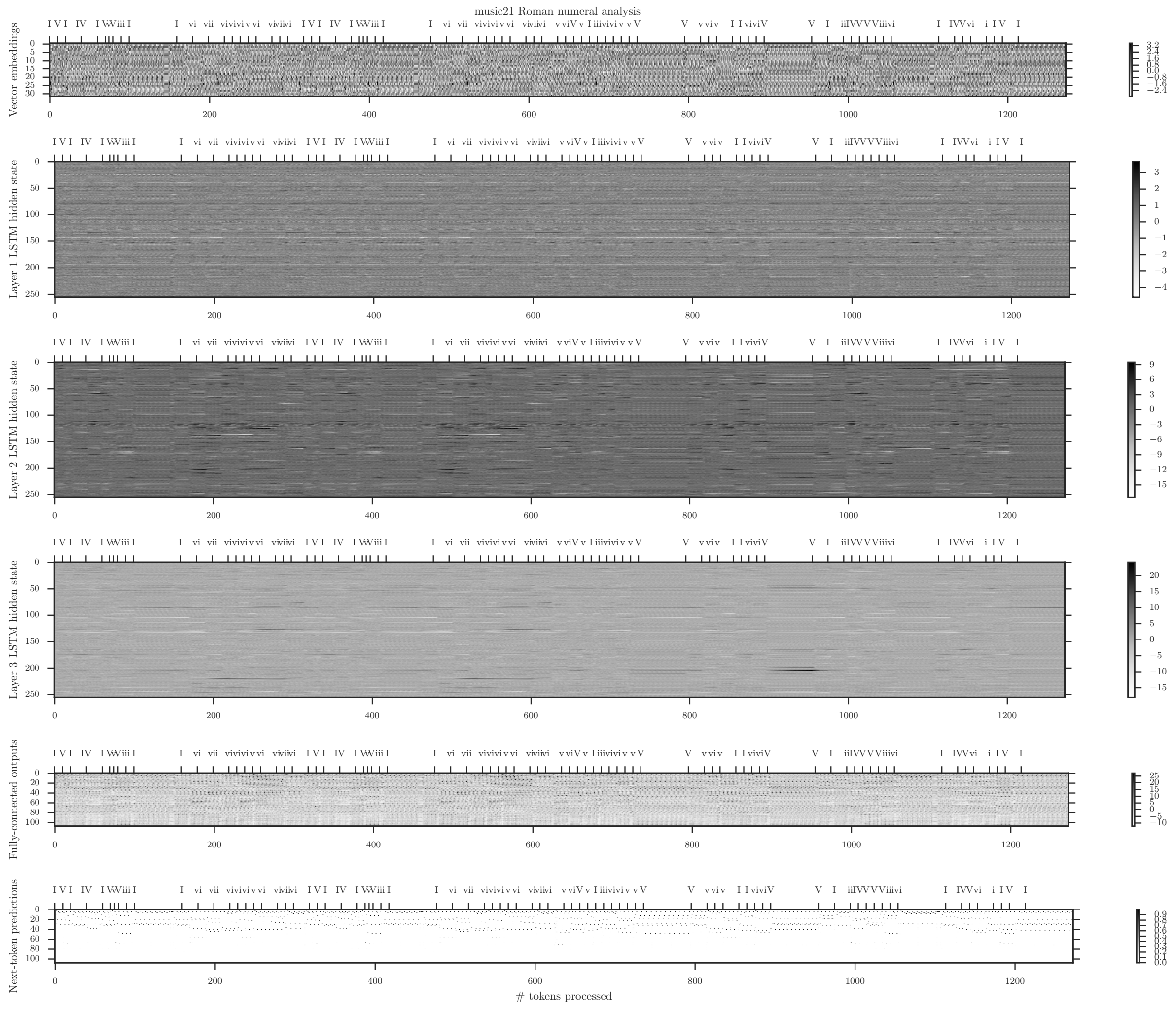


Fig. A.4 Neuron activations over time as the encoded stimulus is processed token-by-token

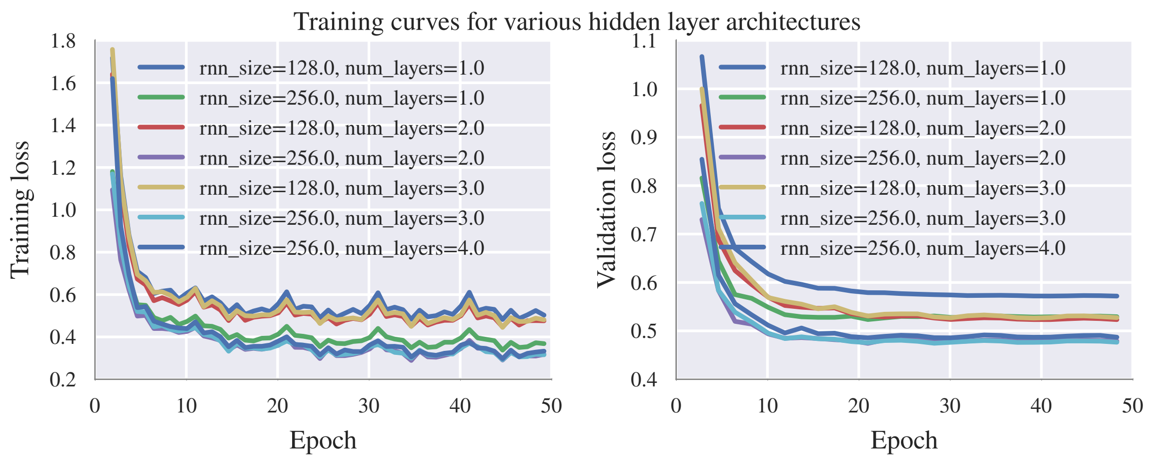


Fig. A.6 `rnn_size=256` and `num_layers=3` yields lowest validation loss.

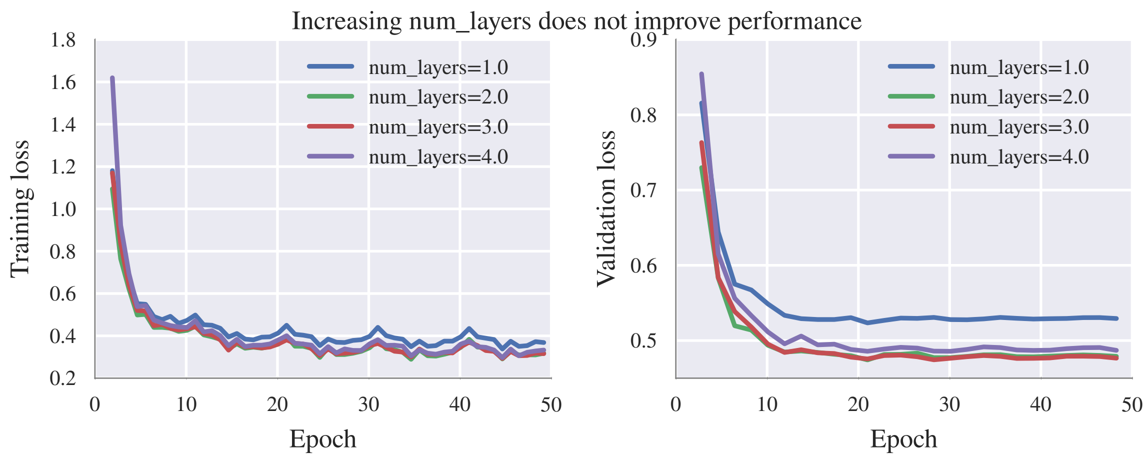


Fig. A.7 Validation loss improves initially with increasing network depth but deteriorates after > 3 layers.

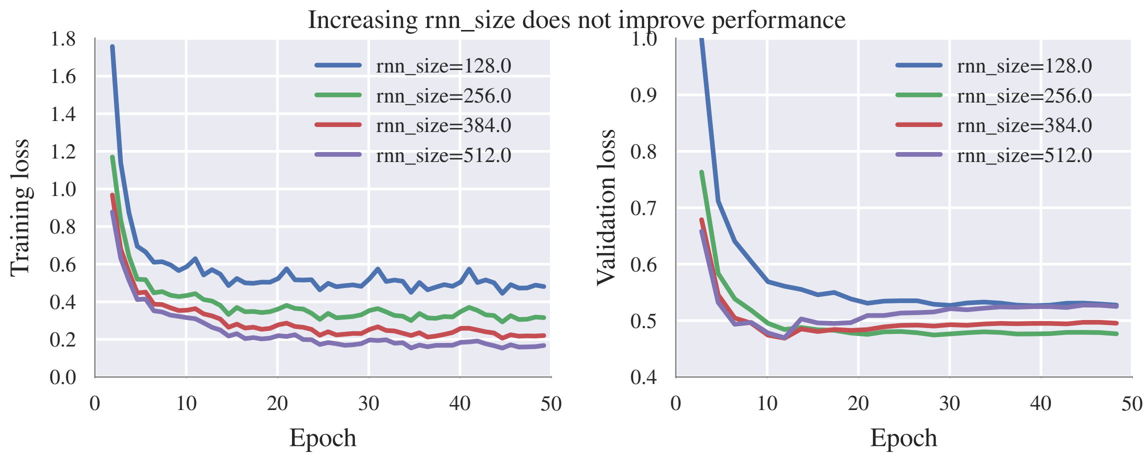


Fig. A.8 Validation loss improves initially with higher-dimensional hidden states but deteriorates after > 256 dimensions.

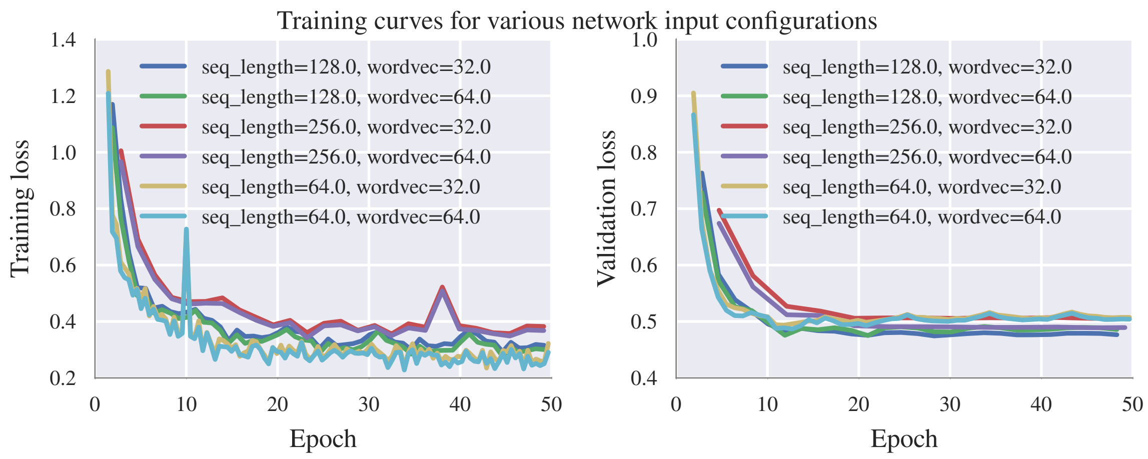


Fig. A.9 `seq_length=128` and `wordvec=32` yields lowest validation loss.

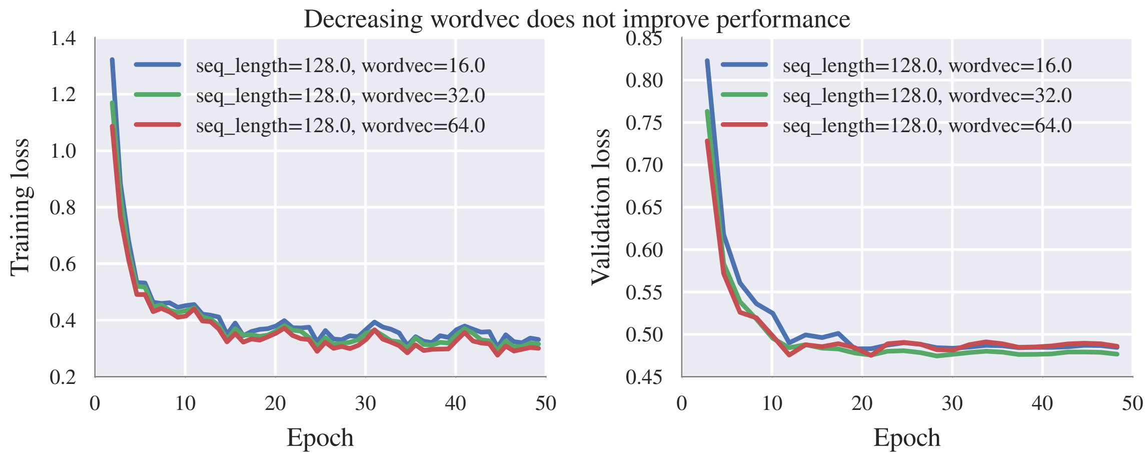
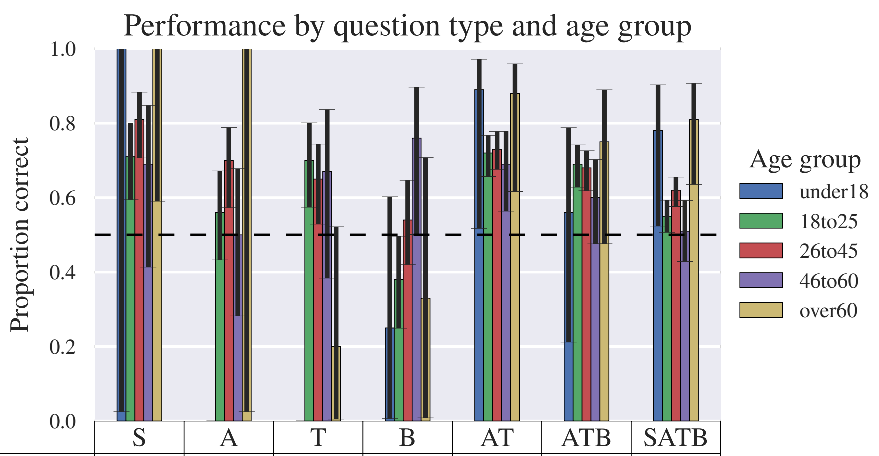


Fig. A.10 Perturbations about `wordvec=32` do not yield significant improvements.



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

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