1 Automatic composition

fliang: Debug minitoc

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

1.1 Constructing a corpus of encoded scores

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

1. The Baroque style employed in Bach chorales has specific guidelines [23] (i.e. no parallel fifths) and stylistic elements (i.e. voice leading) which can be use to qualitatively evaluate success

2. The structure of chorales are regular: all chorales have four parts and consist of a melody in the Soprano part harmonized by the Alto, Tenor, and Bass parts. Additionally, each

chorale consists of a series of *phrases*: groupings of consecutive notes into a unit that has complete musical sense of its own[21]. It is well known

fliang: citep

that Bach denoted ends of phrases with fermatas

fliang: refer back to background

.

3. The Bach chorales have become a standardized corpus routinely studied by aspiring music theorists[25]

The Bach chorales, indexed by the Bach-Werke-Verzeichnis (BWV) numbering system[8], are conveniently provided by `music21`[9].

1.1.1 Preprocessing

Motivated by the transposition invariance of music and prior practice [20] [11] [13] [14], we first perform *key normalization*. The key signature of each score were firsanalyzed using the Krumhansl Schmuckler key-finding algorithm [17] and then transposed according to

fliang: Table XYZ

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

Next, we perform *time quantization* by aligning note start and end times to the nearest multiple of some minimum duration. Our model uses a minimum duration of one 1/16th note, exceeding the time resoltuions of [6] [11] by 2x, [16] by 4x, and [2] by 8x.

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

An example of the distortion introduced through of our preprocessing steps is provided in 1.1 on the facing page in sheet music notation and in piano roll

fliang: Is piano roll defined?

notation on 1.2 on page 10.

Corpus level analysis of preprocessing effects

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

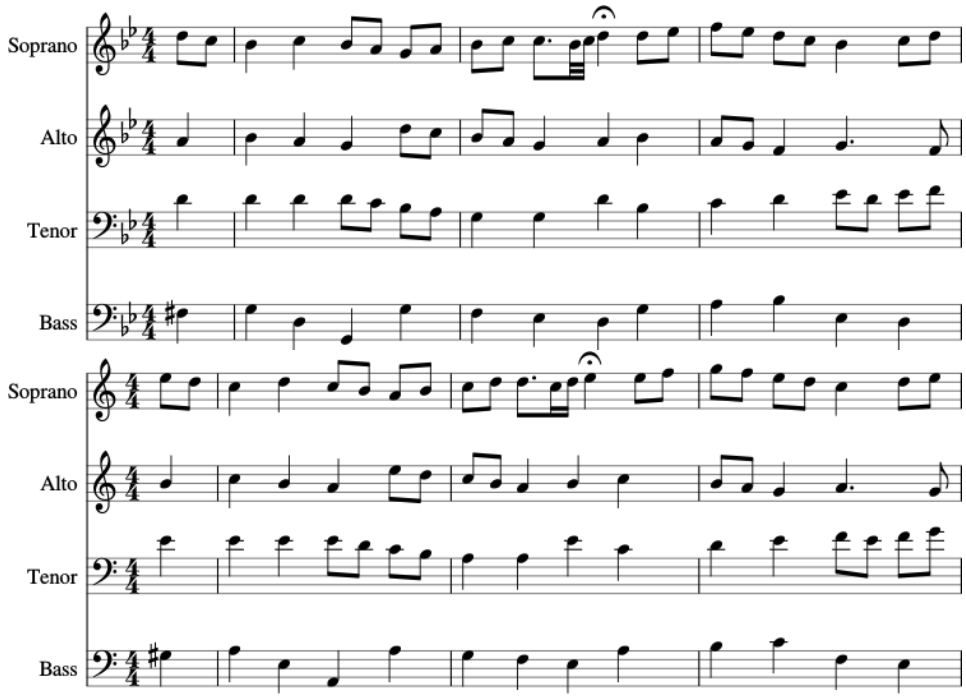

Fig. 1.1 First 4 bars of JCB Chorale BWV 133.6 before (top) and after (bottom) preprocessing. Note the transposition down by a semitone to C-major as well as quantization of the demisemiquavers in the third bar of the Soprano part.

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

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

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

fliang: Update plots... are they affected

.

1.1.2 Sequential encoding of musical data

After preprocessing of the scores, our next step is to encode music into a format amenable for sequential processing. Similar to [24], we represent polyphonic scores using a localist frame-based representation where time is discretized into constant timestep *frames*. Frame based processing forces the network to learn the relative duration of notes, a counting and timing task which [15] demonstred LSTM is capable of. Consecutive frames are separated by a unique delimiter ("|||"' in

fliang: Figure of score encoded in text

).

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

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

fliang: reference

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

fliang: Figure of encoded score

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

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

fliang: Figure

) and end symbol ("END" in

fliang: Figure

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

Observe that our encoding is sparse: unarticulated notes are not encoded. It is also variable length as anywhere from zero to four (in the case of chorales, more for arbitrary polyphonic scores) notes. Finally, the explicit representation of tied notes vs articulated notes solves the problem plaguing [11][10] [19] [7] where multiple articulations at the same pitch are indistinguishable from a single note with the same duration.

Additionally, notice that our encoding avoids hand-engineered features such as pitch representations which are psychochologically-based [20] or harmonically-based [13] [18]. This is intentional and is motivated by numerous reports [4][5] suggesting that that a key ingredient in deep learning's success is its ability to learn good features from raw data.

1.2 Design and validation of a generative model for music

In this section, we describe the design and validation process leading to our generative model. Unlike many prior models for music data, we intentially avoid injection of domain-specific knowledge into our model architectures such as distinguishing between chords versus notes [16][20] [11] and explicitly modelling of meter [10] or motifs [12]. Through this fundamental connectionist approach, we aim to minimize biases introduced by prior assumptions and force the model itself to learn musical structure from data.

1.2.1 Training and evaluation criteria

Following [20], we will train the model to predict a distribution distribution over all possible tokens next $\boldsymbol{x}_{t+1}$ given the current token $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$. This is equivalent to setting the target sequence to be the input sequence delayed by one timestep: $\boldsymbol{y}_{1:T-1} = \boldsymbol{x}_{2:T}$ and $\boldsymbol{y}_T = \text{STOP}$.

fliang: Diagram for sequential prediction

.

fliang: Note similarity with language modeling

.

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

Note that our training criteria as written in

fliang: reference

uses the actual next token $\boldsymbol{x}_{t+1}$ as the recurrent input, even if the most likely prediction argmax $P(\boldsymbol{x}_{t+1}|\boldsymbol{h}_t, \boldsymbol{x}_t)$ differs. This is is referred to as *teacher forcing*[26] and is motivated by the observation that model predictions may not yet be correct during the early iterations of training. However, at inference the token generated from $P(\boldsymbol{x}_{t+1}|\boldsymbol{h}_t, \boldsymbol{x}_t)$ is reused as the previous input, creating a discrepancy between training and inference. Scheduled sampling [3] 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.

1.2.2 Comparing memory cells for music data

Using theanets

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

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

90/10 train/test split.

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

LSTMs and GRUs trained much faster and achieved lower training loss, suggesting higher capacity. [22] reports that LSTMs outperform GRUs in music applications, motivating our final choice for GRUs.

1.2.3 Optimizing the LSTM architecture

Switched to `torch-rnn`.

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

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

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

fliang: citep

.

Models were trained using RMSProp

fliang: citep

with batch normalization

fliang: citep

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

fliang: citep Mikolov

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

Overall best model

We identified our best model to be

fliang: what is it?

.

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

Regularization

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

fliang: Batch normalization experiments

Network capacity

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

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

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

- Training curves about the same wrt `num_layers`, validation curves have interesting story
  - Depth matters: small 64 and 128 hidden unit RNNs saw improvements up to 0.09
  - Expressivity gained from depth furthers overfitting: 256 hidden unit RNN has some of the best validation performance at depth 1 but is the worst generalizing model for depths 2 and 3 even though training loss is low
- `rnn_size=128` undisputably best generalizing, optimized at `num_layers=2`: will continue with these settings

Network input parameters

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

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

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

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

1.3 Results

fliang: Compare with [1] and [7]

.

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

1.4 Accelerating model training with GPUs

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

1.5 Other applications

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

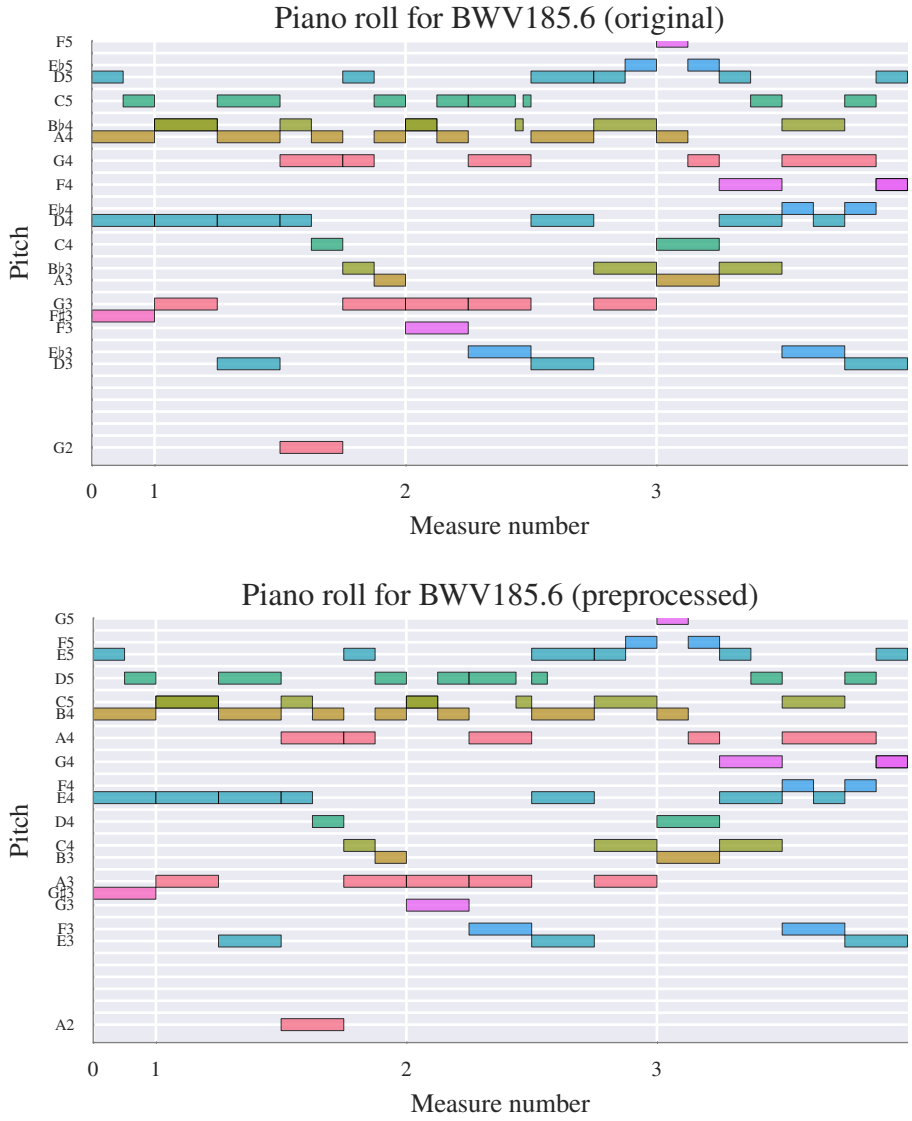

Fig. 1.2 Piano roll representation of the same 4 bars from **??** before and after preprocessing. Notice the transposition of key as well as quantization of the C5 and D5 notes (after transposition) in bar 3 of the Soprano part.

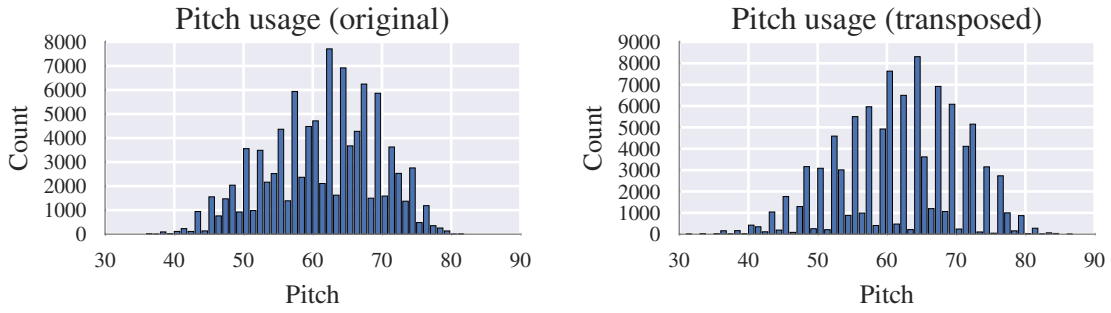

Fig. 1.3 Pitches before and after key standardization

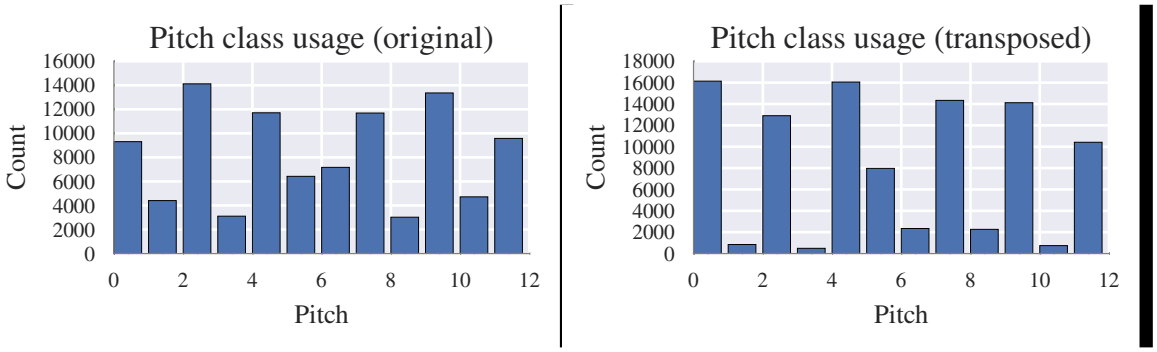

Fig. 1.4 Pitch classes before and after key standardization

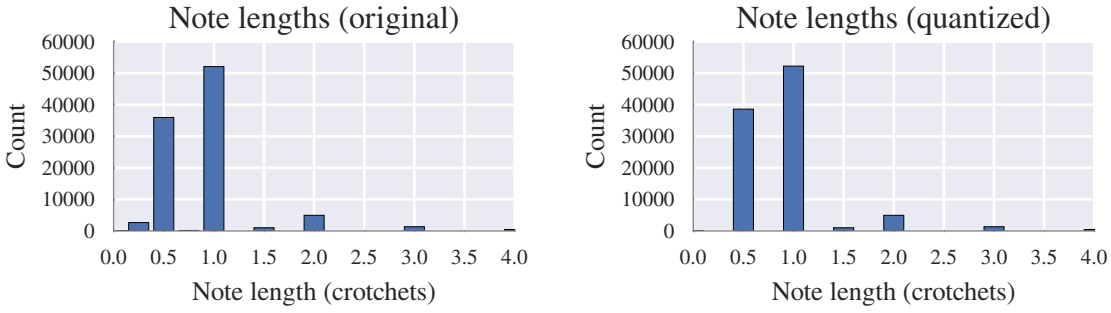

Fig. 1.5 Effects of time quantization on note durations

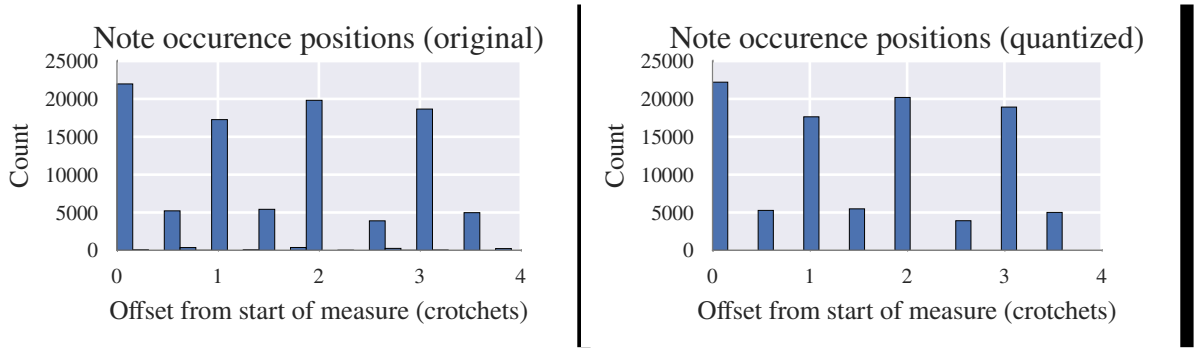

Fig. 1.6 Effects of time quantization on meter

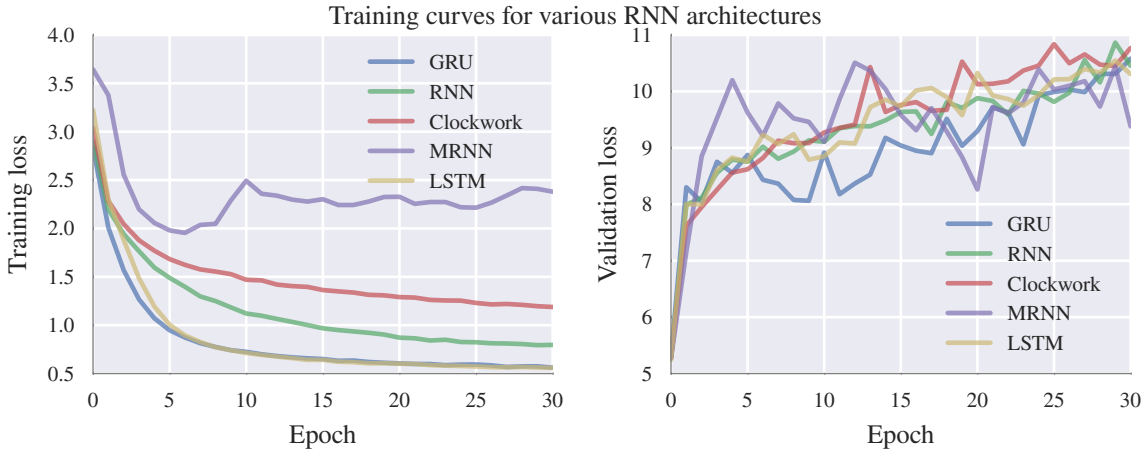

Fig. 1.7 theanets-architecture

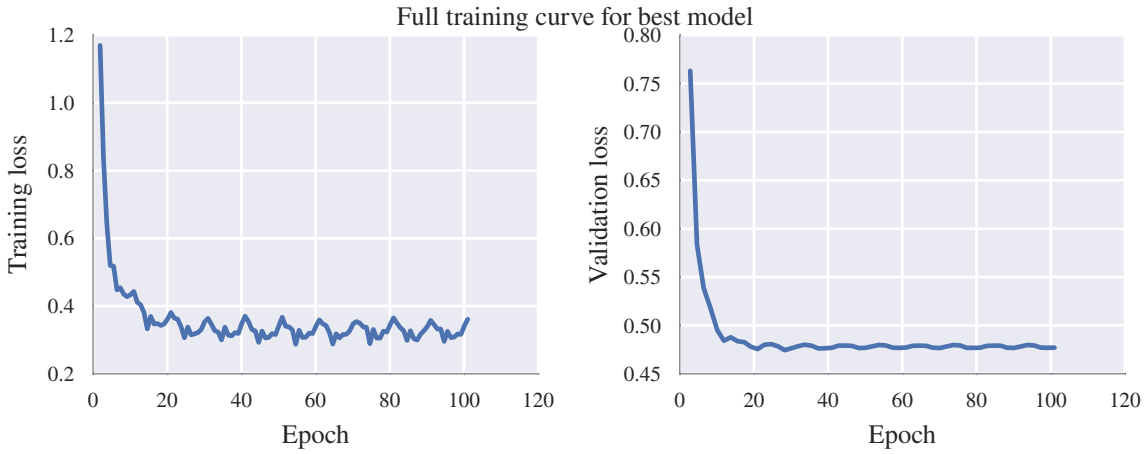

Fig. 1.8 torch-rnn-best-model-Trace

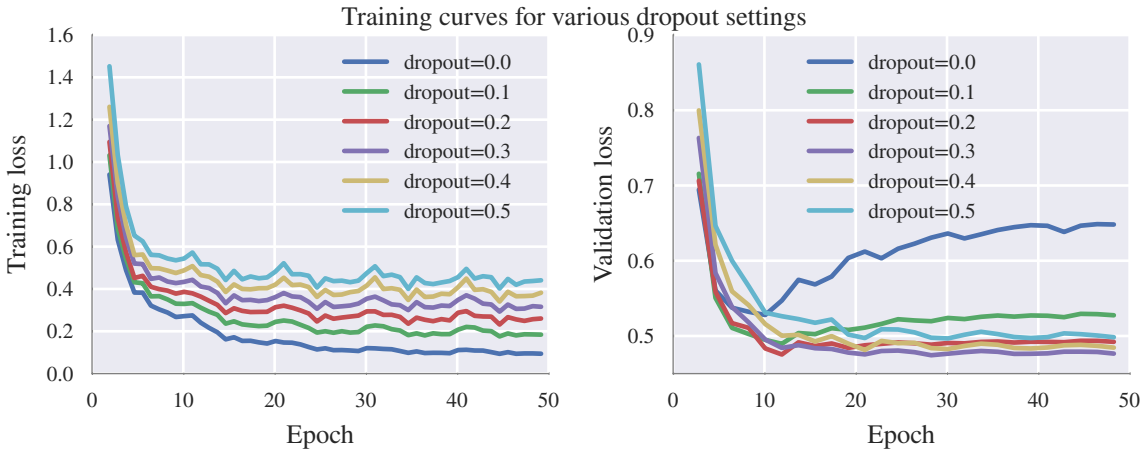

Fig. 1.9 torch-rnn-Dropout

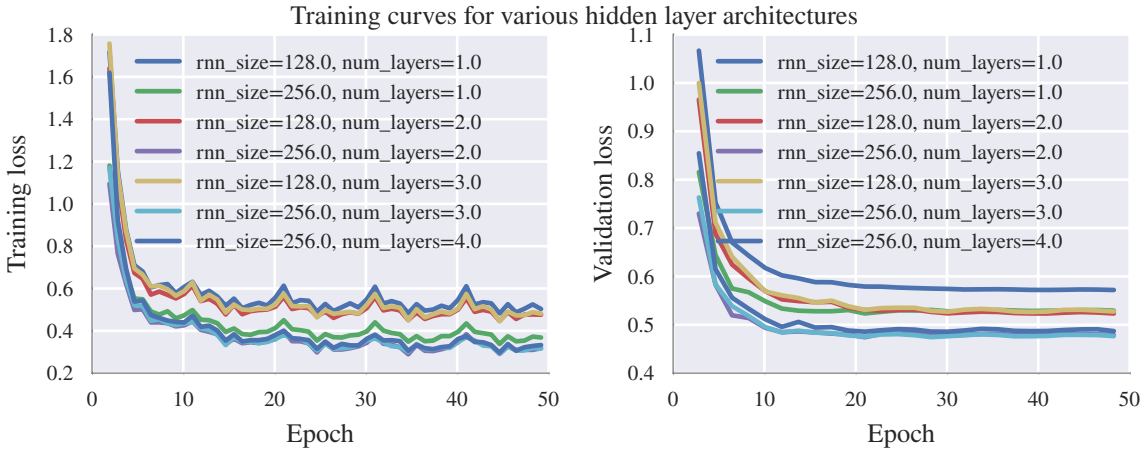

Fig. 1.10 torch-rnn-network-params

Fig. 1.11 torch-rnn-network-params-num-Layers

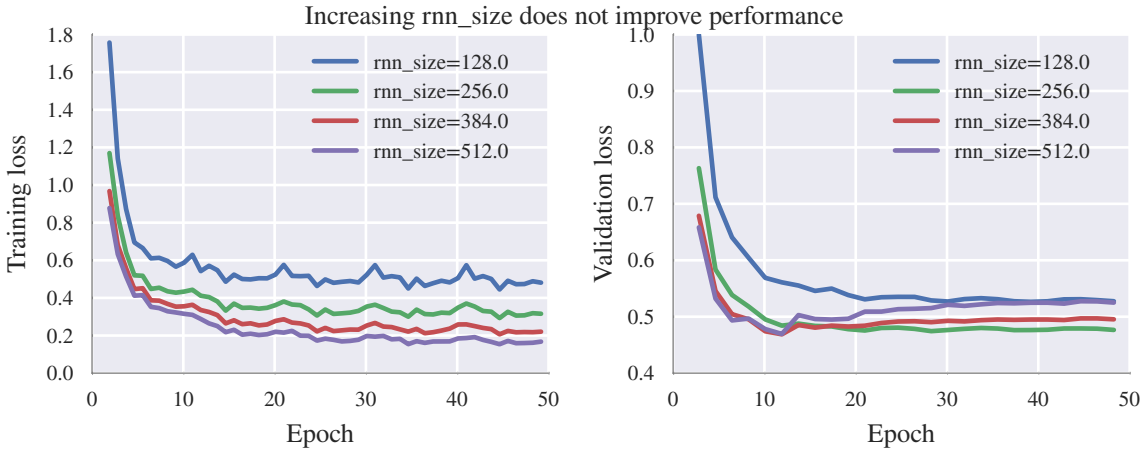

Fig. 1.12 torch-rnn-network-params-rnn-size

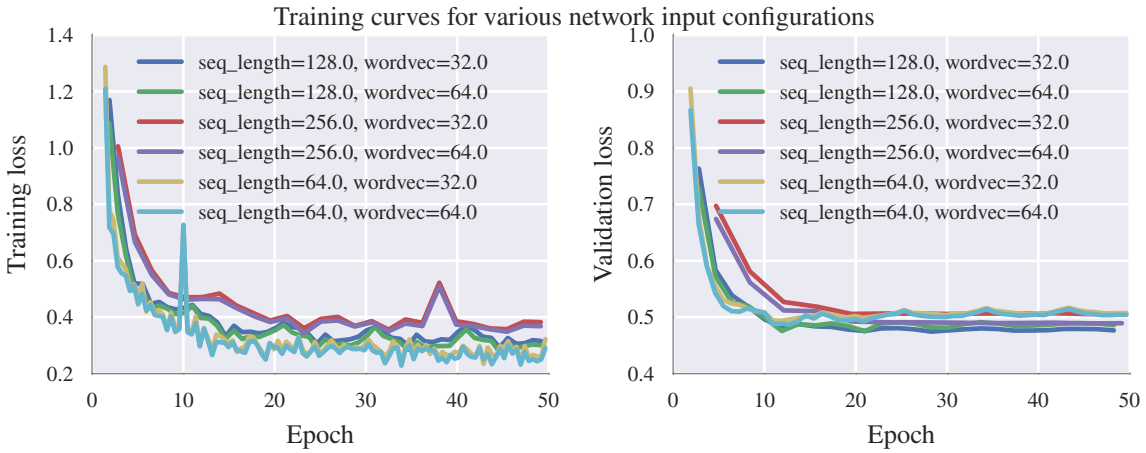

Fig. 1.13 torch-rnn-input-params

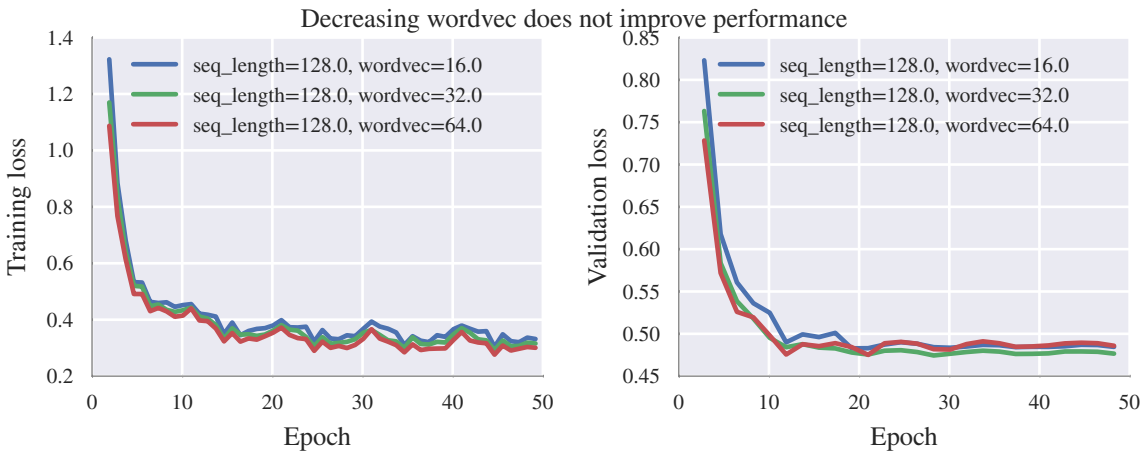

Fig. 1.14 torch-rnn-input-params-Wordvec

References

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

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

[3] Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179.

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

[5] Bengio, Y. and Delalleau, O. (2011). On the expressive power of deep architectures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6925 LNAI:18–36.

[6] Boulanger-Lewandowski, N., Vincent, P., and Bengio, Y. (2012). Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription. *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, (Cd):1159–1166.

[7] Brien, T. O. and Roman, I. (2016). A Recurrent Neural Network for Musical Structure Processing and Expectation. *CS224d: Deep Learning for Natural Language Processing Final Projects*, pages 1–9.

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

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

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

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

[12] Feulner, J. and Hörnel, D. (1994). Melonet: Neural networks that learn harmony-based melodic variations. In *Proceedings of the International Computer Music Conference*, pages 121–121. INTERNATIONAL COMPUTER MUSIC ACCOCIATION.

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

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

[15] Gers, F. A., Schraudolph, N. N., and Schmidhuber, J. (2002). Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143.

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

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

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

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

[20] Mozer, M. C. (1994). Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2-3):247–280.

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

[22] Nayebi, A. and Vitelli, M. (2015). GRUV: Algorithmic Music Generation using Recurrent Neural Networks. *CS224d: Deep Learning for Natural Language Processing Final Projects*, pages 1–6.

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

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

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

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