A Recurrent Neural Network for Musical Structure Processing and Expectation

Tim O'Brien
CCRMA
Stanford University
Stanford, CA 94305
tsobrien@stanford.edu

Irán Román
CCRMA
Stanford University
Stanford, CA 94305
iran@stanford.edu

Abstract

Research in cognitive neuroscience has identified neural activity correlated with subjects hearing an unexpected event in a musical sequence. However, there is a lack of models attempting to explain how these computations are carried out in the human brain. Using an augmented data set consisting of music from the western tradition (originally 371 Bach chorales), we trained a Long Short-Term Memory (LSTM) and two Recurrent Neural Network (RNN) architectures to ask: will a neural network show a larger perplexity when presented with an unexpected event in a musical sequence? Could this higher perplexity highlight similar computations carried out by the human brain? Our results indicate that our methods to train different neural network architectures with a corpus of music were effective. Perplexity values for the LSTM architecture reached low values in the order of 50. Visualization of the TSNE-reduced chord embedding matrix showed separation toward the edges common chords in our vocabulary compared to less common ones. Moreover, perplexity for an unexpected chord in an otherwise well-formed chord sequence was larger compared to an expected chord, but not as large as we originally hypothesized. Finally, out of all the architectures we trained, the clockwork RNN was the one less prone to overfitting the data, and the one that generated the most sensical chord progressions after training. Our results show a promising path moving toward the development of a neural network model of musical structure processing and expectation.[1]

1 Introduction

Similar to language, music is a universal element of the human condition. Because of its organized nature, all humans exposed to music implicitly learn the hierarchical dependencies connecting events in music, even if they do not explicitly receive formal musical training [1]. Research in neuroscience shows that all humans are able to identify events that violate common structures in music. Recently, Román and Fujioka found evidence showing that humans with different levels of musical training group musical sequences in structures that follow or violate local and global dependencies [2]. However, there is a lack of models explaining the underlying computations taking place in the brain when humans listen to sound events unfolding in a musical sequence. Our goal is to develop such a model explaining how humans analyze unexpected events in music which elicit surprise. We use the term *surprise* to encompass the perceptions of subverted expectation, a sense of musical novelty, or the notion of a "wrong" or out-of-place note or chord.

A recurrent neural network (RNN) can carry out such a task. RNNs can develop expectations in sequences, relying on information about recent events and hence the structural dependencies in mu-

[1] Accompanying code is available at `https://cm-gitlab.stanford.edu/tsob/musicNet/`

sical sequences. Such an approach is similar to language models in the domain of natural language processing. After training, such a model could recognize unexpected events in a musical sequence in a manner similar to humans. Additionally, the network can generate musical sequences after training, demonstrating the abstractions it has learned from the data on which it was trained.

2 Background

Cognitive neuroscience provides evidence about how humans process sequences of events through time [3], how sensory and cognitive priming lead to the construction of mental representations of musical events [4], and how the life background of an individual affects expectation when listening to a sequence of musical events [5]. Inspired by this evidence, Berger and Gang [6, 7, 8, 9] hypothesized and developed a computational model consisting of simple recurrent networks, among others, able to parse musical sequences of events and develop expectations after training.

More recently, Rabovsky [10] revisited the work by McClelland *et al.* [11] on computational modelling of language cognition. Rabovsky found that McClelland's model, similarly to other models by Frank [12] for semantic processing, feature large activations in the second hidden layer when presented with semantic violations. Such semantic violations are also known to elicit specific patterns of brain activity in humans, as measured by EEG [10].

Our interests are more aligned with syntax (structure) processing than semantics and thus we believe that Rabovsky's architecture will not serve our purposes. Instead an RNN, similar to and building upon the work by Berger and Gang, will be our architecture of choice. Thus, considering the structural similarities between music and language, we built a language model trained on musical structures that shows larger perplexity when stimulated with unexpected events that violate the syntactic rules underlying musical flow.

3 Approach

The methods described here allowed us to train an RNN on a corpus of music that consists of sequences of chords. Our data is analogous in structure to language, since chords are discrete events, similar to words, and sequences of chords form musical phrases, similar to how words in language form sentences. Thus, the dependencies between simultaneous chords in our data-set will be captured by the weights in the hidden modules of our Neural Network architectures.

3.1 Data Set

We obtained a data-set consisting of all four-voice chorales by Johann Sebastian Bach in the Hum-Drum digital format. This corpus of music contains musical features that are common in western music, and should capture the musical abstractions that individuals in the western world are exposed to since an early age. Bach was an iconic composer from the 18th century. He wrote pieces that set the stylistic foundations for composers of classical music. Among Bach's oeuvre, his chorales (371 pieces in four voices: bass, tenor, alto, and soprano) are an influential body of music sharing multiple features with western music from other composers and other eras. Bach chorales consist of four different but simultaneously sounding voices. These voices coexist in separate pitch ranges (bass, tenor, alto, soprano). For en example of this musical texture, see the fragment in figure 1.

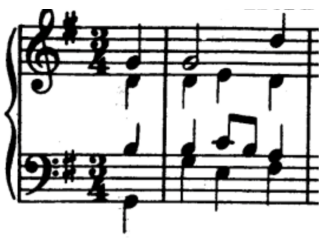


Figure 1: The beginning of a Bach chorale.

3.2 Data Preparation

In western music there are 12 different pitch classes within an octave. We parsed Bach chorales by concatenating four octaves (one for each voice) in a 48-dimensional vector (12 dimensions for each of the voices). Thus, a 48-dimensional row vector corresponds to the sonority produced by the four simultaneously sounding voices at a point in time. Since rhythm in music can make these chords sound repeatedly for an extended period of time, repeated row vectors indicate a chord lasting longer than the shortest unit of musical subdivision we chose, which turned out to be an eighth note. Fig. 2 shows our parsed representation of the fragment from Fig. 1.

Our original data set only contained 370 chorales, all in the same key of C major. In order to expand our data set, we used a standard method from music theory to augment our musical material called "transposition." Transposing a piece changes the reference pitch that defines the "tonic" of the piece. After carrying out data augmentation, all twelve pitches in the set served as tonic for each of our chorales. This data augmentation procedure expanded our data set by a factor of 11.

Thus, in both our original and augmented data-sets, a "word" is each of the time-step vectors (48 dimensional containing all voices sounding simultaneously), and a "sentence" is a sequence of these row vectors unfolding over time to form an entire chorale.

Bach chorale example

t=1	1,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,1,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,1,0,0,0,0, 1,0,0,0,0,0,0,0,0,0,0,0
t=2	1,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,1,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,1,0,0,0,0, 1,0,0,0,0,0,0,0,0,0,0,0
t=3	1,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,1,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,1,0,0,0,0, 1,0,0,0,0,0,0,0,0,0,0,0
t=4	1,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,1,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,1,0,0,0,0, 1,0,0,0,0,0,0,0,0,0,0,0
t=5	0,0,0,0,0,0,0,0,1,0,0, 0,0,0,0,0,1,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,1,0,0, 1,0,0,0,0,0,0,0,0,0,0,0
t=6	0,0,0,0,0,0,0,0,0,1,0,0, 0,0,0,0,1,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,1,0,0, 1,0,0,0,0,0,0,0,0,0,0,0
t=7	0,0,0,0,0,0,0,0,0,0,0,1, 0,0,1,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,1,0,0,0,0, 0,0,0,0,0,0,0,0,1,0,0,0
t=8	0,0,0,0,0,0,0,0,0,0,0,1, 0,0,1,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,1,0,0,0,0, 0,0,0,0,0,0,0,0,1,0,0,0

Figure 2: The same Bach chorale snippet as in Fig. 1, encoded as concatenated length-12 row vectors for each time-step, corresponding to the duration of an eighth note. The groups of 12 numbers represent, from left to right, the bass, tenor, alto, and soprano.

3.3 Neural Network Architectures

Our experiments focused on two flavors of feed-forward recurrent neural networks, the LSTM (see §3.3.1) and the Clockwork RNN (see §3.3.2). In both architectures, we take each chord (*i.e.* the "word" as described in §3.2) and learn an embedding matrix such that each unique chord is represented as a vector in our input space.

3.3.1 LSTM

We use a multi-layer LSTM architecture. For practical computational reasons, our experiments consisted of versions with two LSTM layers, as is depicted in Figure 3. For each LSTM layer, we may characterize the computation by the following equations. Equations 1-4 define what are known as the input gate $i^{(t)}$, forget gate $f^{(t)}$, output gate $o^{(t)}$, and new memory cell $\tilde{c}^{(t)}$.

$$i^{(t)} = \sigma\left(W^{(i)}x^{(t)} + U^{(i)}h^{(t-1)}\right) \tag{1}$$

$$f^{(t)} = \sigma\left(W^{(f)}x^{(t)} + U^{(f)}h^{(t-1)}\right) \tag{2}$$

$$o^{(t)} = \sigma\left(W^{(o)}x^{(t)} + U^{(o)}h^{(t-1)}\right) \tag{3}$$

$$\tilde{c}^{(t)} = \tanh\left(W^{(c)}x^{(t)} + U^{(c)}h^{(t-1)}\right) \tag{4}$$

Note that σ denotes the sigmoid function, $\sigma(z) = (1 + e^{-z})^{-1}$. The superscripts $^{(t)}$ denote the index of the current time step. Using the above equations, we can compute our final memory cell,

$$c^{(t)} = f^{(t)} \circ \tilde{c}^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}\ , \tag{5}$$

where $\circ$ denotes the Hadamard product. Finally, our hidden state is

$$h^{(t)} = o^{(t)} \circ \tanh\left(c^{(t)}\right)\ . \tag{6}$$

The new memory $\tilde{c}^{(t)}$ and hidden state $h^{(t)}$ are passed along for use in computing the subsequent time step $(t+1)$ values, and the hidden state $h^{(t)}$ is passed forward to serve as input to the next layer in the current time step t (*i.e.* the next LSTM layer or the output layer).

Thus, there are eight learned matrices in each LSTM layer: four each of the $W^{(\cdot)}$ and $U^{(\cdot)}$ matrices in Equations 1-4. For our implementation, we initialize these matrices with a uniform random zero-mean distribution. The size of the distribution (*i.e.* the maximum absolute value of any element) is a tunable hyperparameter.

The system includes gradient norm clipping in order to prevent exploding gradients. Gradient norm clipping was first introduced by Pascanu, Mikolov, and Bengio [13], and prevents exploding gradients (and thus exploding stochastic gradient descent training) by clipping gradients whose norms exceed a certain threshold. This threshold in our case is a tunable hyperparameter.

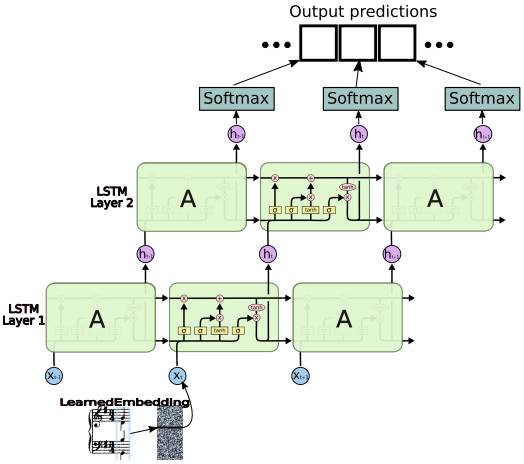


Figure 3: The two-tiered LSTM architecture.[a]

[a]Note that the LSTM layers in the diagram were taken from Christopher Olah's blog post, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

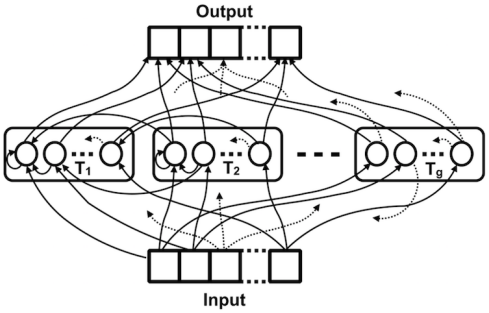


Figure 4: The clockwork RNN has g hidden layers with different time steps T_i. In this figure faster time steps are on the left (T_1, T_2) and slower ones are on the right (T_g). Notice that hidden layers with slower time steps are connected to faster time steps, but not vice-versa.

3.3.2 Clockwork RNN

Another neural network architecture we trained was the clockwork RNN (cwRNN) [14]. The cwRNN is in essence a standard RNN, but has hidden layers whose weight matrices get updated at different time steps. A diagram of this model is depicted in figure 4. The different time steps for each hidden layer allow this architecture to learn both long-term and short-term dependencies in the sequential data that it is trained on. Moreover, in the cwRNN a given hidden layer with time step T_i receives information from another hidden layer with time step T_g if $T_i < T_g$. This allows faster hidden layers to capture information about the large-scale dependencies in the data that slower hidden layers learn.

The computation of the state in this cwRNN at every time-step is given by the expression:

$$y_H^{(t)} = f_H(W_H \cdot y^{(t-1)} + W_I \cdot x^{(t)})$$

$$y_O^{(t)} = f_O(W_O \cdot y_H^{(t-1)})$$

where the subindices I, H, and O, indicate elements in the input, hidden, and output layers. $x^(t)$ is the input of the network, and f describes activation functions for the hidden and output layers of the network. However, only the rows and columns in W_I and W_H for hidden layers that suffice

$t\ MOD\ T_i = 0$ get updated at time step t. Thus, the matrices W_I and W_H have g rows, but matrix W_H is a block-upper triangular matrix with

$$W_H = \begin{cases} W_{Hi} & \text{for } (t\ MOD\ T_i) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The implementation of our clockwork RNN, is a modification of the text-specific cwRNN model in the `theanets` package.[2] The `theanets` implementation of this cwRNN is an implementation following the description in the original paper [14]. Our implementation differs mainly from the original in that the text was originally parsed so that each letter corresponds to a time-step, while our model requires text to be treated on a word-by-word basis, where each sequential chord in our data corresponds to a time step.

4 Experiments and Results

To test our architectures, we carried out different experiments with the LSTM and the clockwork RNN architectures previously described. All code is available at the following git repository: `https://cm-gitlab.stanford.edu/tsob/musicNet/`.

4.1 LSTM

All LSTM experiments were run using TensorFlow 0.8.0 on a machine running Fedora Linux (Planet CCRMA[3]), and utilizing an NVIDIA GeForce GTX 750 (1.2935 GHz) using the cuDNN 4 and CUDA 7.5.18 libraries.

Our best-performing model was trained on the aforementioned 370 four-part Bach chorales. The training set consisted of 70% of this corpus, or 259 chorales; the validation set consisted of another 10%, or 37 chorales; and the test set consisted of the final 20%, or 74 chorales. With data augmentation via transposition of each chorale to each of the twelve possible keys, we have 3108, 444, and 888 chorales for the training, validation, and test sets, respectively.

The initial scale for our trainable tensors was 0.05, which means every element was randomly assigned a value between -0.05 and 0.05. The size of our chord embedding vectors and our hidden states was 400, and the vocabulary size was 10000.

The learning rate, using vanilla minibatch stochastic gradient descent, was 1.0, which decayed by a factor of 0.8 after epoch 6 until the final 39th epoch. Batches of 20 were used. The gradient norm threshold, for clipping, was 5. During training, we applied dropout at a probability of 50%. The training history, plotted in Fig. 5, shows an orderly optimization trajectory.

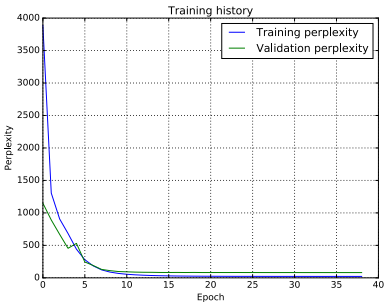


Figure 5: LSTM training history.

We plot a t-SNE representation [15] of the chord embeddings in Fig. 6. All except one of the annotated chord types are triads, and we chose to highlight the most common four-part voicings – namely, those in which the bass note is the root, the root is repeated in an upper voice, and no chord notes are omitted. The dominant G7 chords, on the other hand, include all voicings with a G in the bass, but allow for the omission of the 3rd or 5th in order to double the root in an upper voice and retain the 7th.

For simplicity, we present the embedding matrix from a system trained on non-augmented chorale data, all of which were normalized to have a tonic of C. Thus, for example, the green C majors may tend to represent the major I chord of a chorale, though of course modulations within a chorale were not altered, so the function of C major may be different in a modulated context.

[2] `theanets` is available at thefollowing repository: `https://github.com/lmjohns3/theanets`
[3] `http://ccrma.stanford.edu/planetccrma/software/`

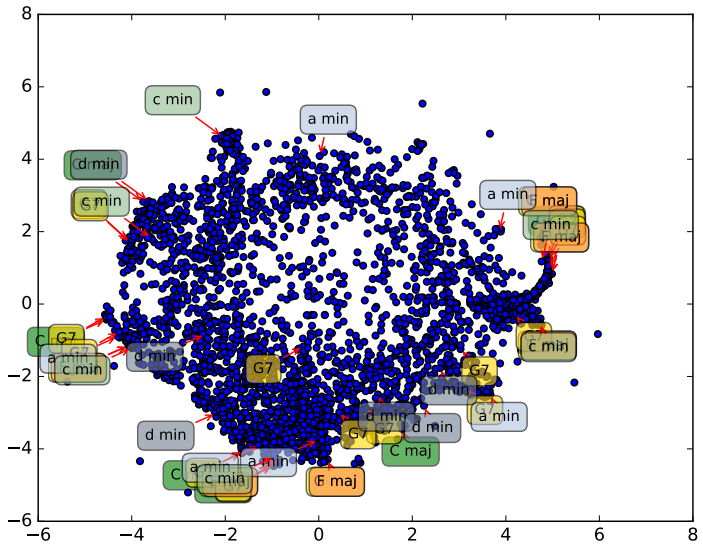
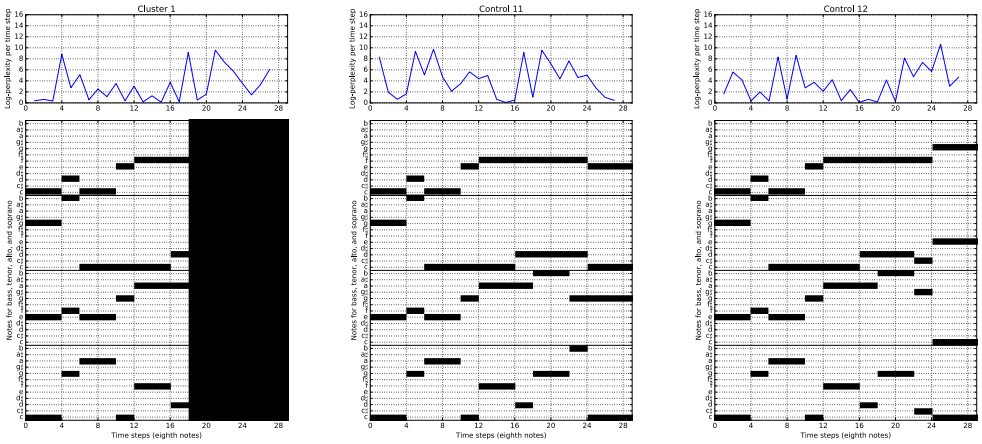


Figure 6: A t-SNE visualization of LSTM chord embeddings, with several chord types labeled.

What strikes us as interesting is that the annotated chords cluster at the edges, though different voicings of the same chord may appear far apart. Voicings of C major and G major or G dominant 7 sometimes appear in close proximity. This was at first surprising, given that the tonic and dominant serve quite distinctly different harmonic functions; on the other hand, tonic and dominant chords are highly likely to appear next to each other in sequence, for example at cadences or to establish key.



(a) A very unexpected cacophony interrupts a normal progression.

(b) A perfectly normal harmonic progression.

(c) A normal progression with an out-of-key penultimate chord.

Figure 7: Perplexity per time step (top) for three input sequences (bottom, represented as stacked piano rolls for each of the four voices).

We present the responses of our trained model to three different test signals in Fig. 7. Fig. 7a shows the perplexity per eighth-note time step (top signal) for a progression that unfolds normally until an abrupt cacophony of every note at every subsequent time step. This is akin to resting one's arm across all the keys of a piano. As expected, we see higher perplexity in that region of the progression. Interestingly, the perplexity subsides quickly, spikes back up, and then subsides more slowly. We conjecture that this results from the fact that, although this is indeed a surprising chord to hear, the

fact the it repeats every time step is a similar gesture to other, more sensical, chord progressions with drawn-out chords. However, we would like to see a much larger spike in perplexity, corresponding to the jarringly surprising nature of the musical event represented.

Fig. 7b shows a progression which follows all rules of tonal harmony and counterpoint. Thus, we would expect to see minimal perplexity. However, there are some spikes in perplexity similar in magnitude to the previous example. We may note that the third chord is an A minor, serving as a deceptive cadence (since it comes after C major and then the G dominant 7). This is not novel and certainly occurs in the model's training set, but as the name implies, the deceptive cadence continues to subvert expectation and surprise human listeners even though they have heard deceptive cadences many times before. As such, a higher level of model perplexity for that instant is somewhat desirable and "correct."

Finally, Fig. 7c shows a perfectly fine harmonic progression except for the penultimate chord. This chord, known as a Neapolitan, was seldom used by Bach in his chorales. Thus, we would expect a higher level or perplexity around this event, and indeed we see it.

4.2 cwRNN

Using a subset of the augmented data (4000 chords total, separated into 3000 for training and 1000 for validation), we trained three different models: an RNN, an LSTM, and a clockwork RNN to assess their performance on our dataset. After training, all models were able to generate chord progressions, which varied in structural features.

Figure 8: The top panel shows the loss function for the training set across epochs. The bottom panel shows the loss function for the validation set. The clockwork RNN is the one that was less prone to over-fitting the subset of data it was trained with.

All models were able to over-fit the data (see Figure 8), but the clockwork RNN was the one with a lesser tendency to do so, as seen in the loss functions for the validation set. This is consistent with previous findings associated with the clockwork RNN architecture when trained on sequences of events. The clockwork RNN architecture is designed to see beyond short-term dependencies, which are usually the ones that a standard RNN model can capture due to the vanishing gradient problem. In the clockwork RNN, connections from hidden layers with slower time periods to ones with faster ones let this architecture capture relationships in longer time scales.

Finally, we made the cwRNN generate music after it was trained. Figure 10 contains the output for this experiment. The chorale reflects the style of the data on which it was trained. Considering it was trained on the augmented data set, it makes sense that this excerpt modulates among different keys and makes use of sophisticated harmony. These features were not observed when a chorale was generated by the ordinary RNN trained on the same data-set, as shown in Figure 9

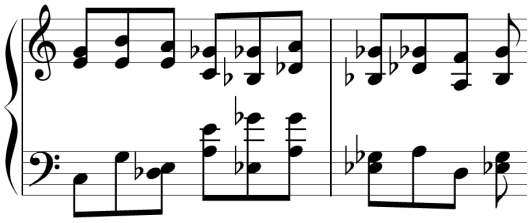


Figure 9: Chord progression generated by an RNN after training. The harmony is incoherent, and seems to move pseudo randomly. There is no clear key and the style and rules of music theory are not followed.

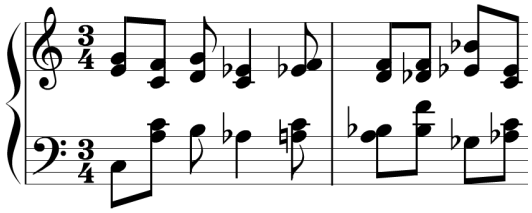


Figure 10: Chord progression generated by the cwRNN after training. The harmony is sophisticated, and follows the rules of music composition and style from the period of the data on which the model was trained.

5 Conclusion

We have seen that recurrent networks, including LSTMs and clockwork RNNs, are suitable for modeling time-sequences of musical data. The language-model approach, in which we use instantaneous prediction error as a proxy for a human listener's surprise, appears to bear fruit, pending further testing and refinement. The clockwork RNN appears to perform best at musical tasks, in resistance to overfit and the generation of musically sophisticated harmonic progressions. This makes sense, as a hallmark of music is the presence of elements and motives varying over different time scales simultaneously.

With our initial reults, as well as the newly available quantitative measures of human listener surprise [2], ideas for future work are manifold and exciting. In addition to prediction error, we would like to look more closely at hidden-layer activations per time step, as they may also yield information relevant to surprise levels.

We hope to continue working on the cwRNN, since it has a musically desirable architecture and generated such interesting music. Beat and rhythm are an emergent property of these models, perhaps especially with the cwRNN; we would like to explore this further, *e.g.* by tuning the cwRNN time scales and/or concatenating explicit rhythmic information to the chord representation.

In the near term, we may scrutinize the embedding matrices more closely to see if there is musically significant local structure. For example, similar to analogies in NLP, perhaps the vector representations of dominant minus tonic are roughly equal. That is, $G7 - C\,maj \simeq A7 - D\,maj$. That and other analogies may help us confirm that the learned embeddings are musically meaningful. Additionally, we may refine the embedding matrix by initializing not randomly, but according to a heuristic such as circle of fifths relationships.

Acknowledgments

We benefited immensely from the advice and assistance of Professors Jonathan Berger and Craig Sapp, of CCRMA and CCARH, respectively, within Stanford University's Department of Music.

References

[1] E. Bigand and B. Poulin-Charronnat, "Are we experienced listeners? a review of the musical capacities that do not depend on formal musical training," *Cognition*, vol. 100, no. 1, pp. 100–130, 2006.

[2] I. Román and T. Fujioka, "Music syntactic processing is influenced by integration of local and global harmonic structures: an erp study.," *Cognition*, 2016.

[3] P. Lalitte and E. Bigand, "Music in the moment? revisiting the effect of large scale structures 1, 2," *Perceptual and motor skills*, vol. 103, no. 3, pp. 811–828, 2006.

[4] E. Bigand, B. Poulin, B. Tillmann, F. Madurell, and D. A. D'Adamo, "Sensory versus cognitive components in harmonic priming.," *Journal of Experimental Psychology: Human perception and performance*, vol. 29, no. 1, p. 159, 2003.

[5] E. Schubert and C. Stevens, "The effect of implied harmony, contour and musical expertise on judgments of similarity of familiar melodies," *Journal of New Music Research*, vol. 35, no. 2, pp. 161–174, 2006.

[6] J. Berger and D. Gang, "A real time model of the formulation and realization of musical expectations." 2010.

[7] D. Gang and J. Berger, "Modeling the Degree of Realized Expectation in Functional Tonal Music: A Study of Perceptiual and Cognitive Modeling Using Neural Networks," in *Proceedings of the International Computer Music Association*, pp. 454–457, 1996.

[8] D. Gang, D. Lehmann, and N. Wagner, "Tuning a Neural Network for Harmonizing Melodies in Real-Time," in *Proceedings of the International Computer Music Association*, pp. 9–16, 1998.

[9] D. Gang and J. Berger, "A unfied neurosymbolic model of the mutual influence of memory, context and prediction of time ordered sequential events during the audition of tonal music," *Hybrid Systems and AI: Modeling, Analysis and Control of Discrete + Continuous Systems*, pp. 53–58, 1999.

[10] M. Rabovsky and K. McRae, "Simulating the N400 ERP component as semantic network error: Insights from a feature-based connectionist attractor model of word meaning," *Cognition*, vol. 132, no. 1, pp. 68–89, 2014.

[11] J. L. Mcclelland, M. St. John, and R. Taraban, "Sentence Comprehension: A Parallel Distributed Processing Approach," vol. 70, pp. 1–54, 1989.

[12] S. L. Frank, L. J. Otten, G. Galli, and G. Vigliocco, "The ERP response to the amount of information conveyed by words in sentences," *Brain and Language*, vol. 140, pp. 1–11, 2015.

[13] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," *arXiv preprint arXiv:1211.5063*, 2012.

[14] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber, "A Clockwork RNN," *arXiv preprint arXiv:1402.3511*, 2014.

[15] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 2579-2605, p. 85, 2008.