GRUV: Algorithmic Music Generation using Recurrent Neural Networks

Aran Nayebi
anayebi@stanford.edu

Matt Vitelli
mvitelli@stanford.edu

Abstract

We compare the performance of two different types of recurrent neural networks (RNNs) for the task of algorithmic music generation, with audio waveforms as input. In particular, we focus on RNNs that have a sophisticated gating mechanism, namely, the Long Short-Term Memory (LSTM) network and the recently introduced Gated Recurrent Unit (GRU). Our results indicate that the generated outputs of the LSTM network were significantly more musically plausible than those of the GRU.

1 Introduction

Algorithmic music generation is a difficult task that has been actively explored in earlier decades. Many common methods for algorithmic music generation consist of constructing carefully engineered musical features and rely on simple generation schemes, such as Markov models or graph-based energy minimization techniques. While these approaches are sometimes able to produce interesting compositions, the resulting musical pieces usually consist of repetitive sequences and lack thematic structures that are common in most musical works.

With the increase in computational resources and recent advancements in recurrent network architectures, novel music generation may now be practical for large-scale corpuses. The most common recurrent network used for modeling long-term dependencies is the Long Short-Term Memory (LSTM) network, introduced by Hochreiter and Schmidhuber [5] in 1997. Recently, Gated Recurrent Units (GRU), introduced by Cho et al. [1], have been used to effectively model long-term dependencies in a variety of generic sequence modeling tasks. We believe that by using LSTM and GRU networks for the task of algorithmic music generation, we can better model the long-term thematic structure of musical pieces and produce compositions that sound unique and musically coherent.

2 Background/Related Work

Recurrent networks have been previously used for music generation and analysis tasks with some success. Eck and Schmidhuber [4] used Long Short Term Memory (LSTM) networks to analyze the structure of common Blues songs using binary vectors to model individual notes appearing in a fixed sequence.

Chung et al. [3] used GRU networks to model polyphonic music sequences using MIDI datasets and found their performance comparable to that of LSTM networks.

While the works listed above have applied recurrent networks to the task of music sequence modeling, neither of these sources attempted the task of recurrent music sequence modeling using raw audio waveforms as inputs. In this sense, our project is entirely novel as our network attempts to perform music sequence modeling by directly operating on the *waveform* representation of the data.

3 Technical Approach and Models

We define our problem of music sequence modeling formally as follows:

Given a set of vectors $X_t, X_{t-1}, \ldots, X_0$, representing audio waveforms at the time intervals t, $t-1, \ldots, 0$, we would like to generate the most likely vector X_{t+1} representing an audio waveform at the next time interval $t+1$.

In essence, this is a sequence modeling task in which we are trying to estimate the next sequence conditioned on the previous sequences. In the case of our problem, the inputs are naturally continuous and hence we can frame the problem of estimating X_{t+1} as a regression task.

We can perform this regression by using a recurrent network in which each feed-forward computation results in a new estimate $\hat{X}_{t+1}$ representing our best guess for X_{t+1}. By training the network on various already known waveforms, we can learn useful priors to aid in future generation tasks.

Our loss function is L2 loss, given by,

$$\ell(\theta) \equiv \frac{1}{T}\sum_t (X_t - \hat{X}_t)^2, \tag{1}$$

where T is the number of timesteps.

Model Representation

One of the benefits of phrasing our problem as a regression task across the waveforms of an audio sequence is that we can feed the recurrent network any manner of audio data we have available for training purposes. As such, all of our training data is represented as raw audio samples extracted from songs encoded in the popular WAV format. In order to simplify training, we convert each song into a mono-channel waveform sampled at 44.1 Khz. Since most songs split audio samples into two channels arbitrarily based on the composer's preference, we would like our algorithm to generalize well across a wide variety of composers and hence, converting the original audio samples into a single channel allows us to compensate for this. It also serves as a means of reducing the dimensionality of our audio data. We fix the sampling rate to 44.1 Khz to allow us to use constant word sizes for training and generation tasks. While audio data is typically represented as samples of a waveform in the time domain, it is usually more meaningful to analyze audio in the frequency domain. After reading the audio samples, we split the audio samples up into blocks of size N and convert each block into its frequency representation using the discrete-Fourier transform (DFT) algorithm. The transformation results in a vector of N real numbers and a vector of N imaginary numbers that collectively make up the phases and magnitudes of the time domain waveform. We concatenate these vectors together to create a $2N$ vector that is used as the internal model representation for our network.

Network Architecture

The two networks we use in our task are the GRU and the LSTM. We describe their architectures below (cf. [3], Sections 3.1-3.2).

GRU architecture

The activation h_t of the GRU at time t is a linear interpolation between the previous activation h_{t-1} and the candidate activation $\tilde{h}_t$:

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t, \tag{2}$$

where the update gate z_t decides how much the unit updates its activation, or content, and $\circ$ denotes element-wise product. The update gate is computed by

$$z_t = \sigma\left(W_z x_t + U_z h_{t-1}\right). \tag{3}$$

The candidate activation is $\tilde{h}_t$ is computed by

$$\tilde{h}_t = \tanh\left(W x_t + r_t \circ (U h_{t-1})\right), \tag{4}$$

and the reset gate r_t is computed by

$$r_t = \sigma\left(W_r x_t + U_r h_{t-1}\right). \tag{5}$$

LSTM architecture

Each LSTM unit maintains a memory c_t at time t. The output h_t is given by

$$h_t = \sigma_t \tanh(c_t), \tag{6}$$

where σ_t is an output gate that modulates the amount of memory content exposure. The output gate is computed by,

$$\sigma_t = \sigma(W_0 x_t + U_0 h_{t-1} + V_0 c_t), \tag{7}$$

where V_0 is a diagonal matrix. The memory cell c_t is updated by partially forgetting the existing memory and adding a new memory content $\tilde{c}_t$:

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t, \tag{8}$$

where the new memory content is

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1}). \tag{9}$$

The extent to which the existing memory is forgotten is modulated by a forget gate f_t, and the degree to which the new memory content is added to the memory cell is modulated by an input gate i_t. Gates are computed by

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + V_f c_{t-1}), \tag{10}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + V_i c_{t-1}), \tag{11}$$

where V_f and V_i are diagonal matrices.

In our implementation, we make use of a single GRU unit and a single LSTM unit to represent transitions between previous hidden states and future hidden states. After a new hidden state has been generated, we use an affine transformation to convert the hidden state directly to its resulting frequency representation. We then evaluate the L2 error between our estimates of the frequency representation and the ground-truth frequency representation.

4 Musical Sequence Generation

After training the network, we can generate new musical compositions by providing a seed sequence $S_t, S_{t-1}, \ldots, S_0$, representing seed audio waveforms at the time intervals t, $t-1, \ldots, 0$. The first step is to perform a feed-forward computation on the seed sequence. After the feed-forward computation is complete, we are left with a new set of vectors $R_{t+1}, R_t, \ldots, R_1$, representing the predicted audio waveforms at the time intervals $t+1, t, \ldots, 1$. We call this collection of vectors the generation sequence. In the second step, we then perform another feed-forward pass with the entire generation sequence as input and append the last vector estimate to the generation sequence. For example, assuming our generation sequence was from $R_{t+k}, R_{t+k-1}, \ldots, R_1$, we would append the estimate for R_{t+k+1} to the generation sequence at the end of the second step.

We then iteratively perform the second step until we have generated a sufficiently long sequence as specified by the user. It is important to note that there are three main parameters to the generation algorithm: the size of the seed sequence, the contents of the seed sequence, and the number of iterations we perform sequence prediction. We discuss these parameters in greater detail in the section below.

5 Experiments & Results

Implementation

We initially wrote our own GRU and LSTM implementations. However, we soon found that these implementations were insufficient for training over a large corpus, mainly due to the fact that they did not leverage GPU acceleration. For instance, as a benchmark, training our GRU implementation on a single 10 second sample of a Madeon song for 100 epochs took 12 hours. As a result, we implemented the GRU and LSTM using the Keras library [2], which uses Theano for fast tensor manipulation and CUDA-based GPU acceleration. We trained both our models on the Stanford Rye clusters, which have powerful GPUs. In particular, the Rye 1 cluster has a Tesla C2070 GPU, and the Rye 2 cluster has a GTX 480 GPU.

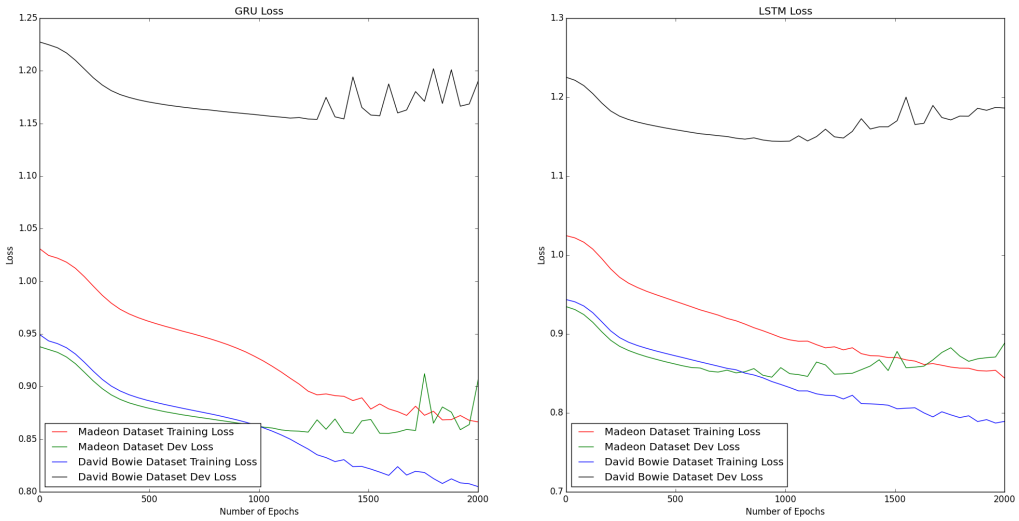

Figure 1: Training and dev loss of the GRU and LSTM for 2,000 epochs on each dataset (Madeon and David Bowie), each having 2048 hidden dimensons.

Datasets

We used two training corpuses, each consisting of twenty songs that are roughly 3 to 7 minutes long from two different artists. The first corpus consisted of 20 popular songs from David Bowie (a rock artist) and the second corpus consisted of 20 popular songs from Madeon (an electronic dance music artist). We also gathered other music from Rock, Pop, EDM (electronic dance music), and Alternative genres, but due to time and memory constraints of our GPU hardware, we were unable to perform thorough training across all genres.

Network Training

We trained both the LSTM and GRU networks on 80% of each corpus with 10 second long clips per training example and performed validation on the remaining 20%. Our intuition was that training separate networks for each corpus likely results in better performance and more coherent melodies than training a single network across all genres. As such, we trained two separate networks for each of the two genres. The LSTM and the GRU networks were both trained for 2000 epochs for each corpus. Each sample is a quarter of a second long and each network consisted of 2048 hidden dimensions. As is evident from Figure 1, the David Bowie dataset exhibits a considerably higher dev loss compared to the Madeon dataset and appears to overfit far sooner as well. We believe the reasons for this are primarily that the David Bowie dataset contains more variety and musical complexity compared to Madeon. In addition, all of Madeon's music is digitally produced, so it is in essence entirely noise-free.

We tested our network under a variety of scenarios to find the optimal hyperparameters for our network. The primary hyperparameters are the number of hidden dimensions, the size of the inputs of the waveforms, and the number of recurrent layers in the network. Overall, we found that adding more recurrent layers made our network much more difficult to train and the weights for both GRU and LSTM were often more likely to become stuck in a local minima early in the training process. The number of hidden dimensions has a direct impact on the quality of our generation and in essence acts like a compression factor for our waveforms. As can be seen in Figure 2, for both the GRU and LSTM, the dev set error is smaller for a larger number of hidden dimensions. Interestingly enough, having hidden dimensions of size 2048 (corresponding to roughly a tenth of the size of our input waveforms) seemed to produce the best results and provided a reasonable compromise between our GPU memory footprint and training time. This hints that audio waveforms are extremely compress-

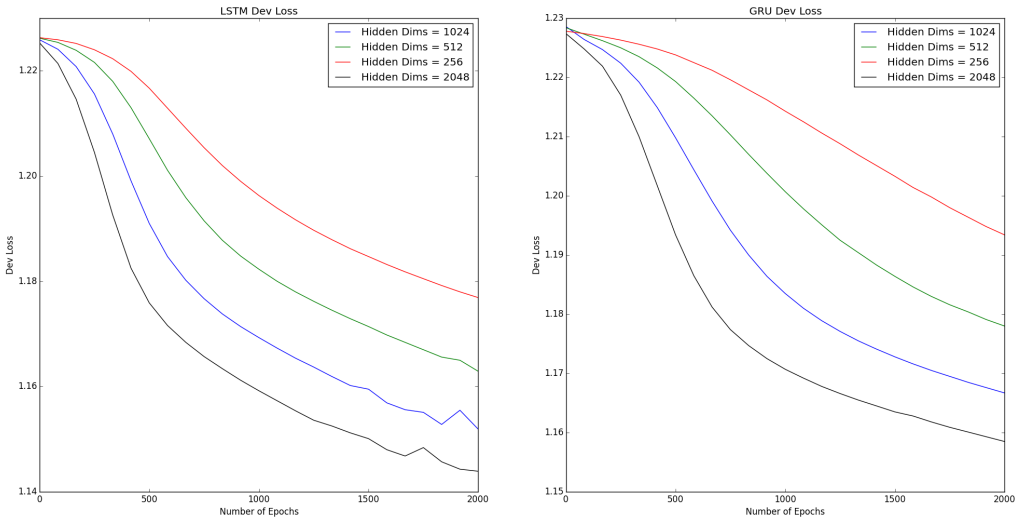

Figure 2: Comparison of hidden dimension sizes on dev set error for both the GRU and LSTM, when trained for 2,000 epochs.

ible and only a subset of the frequencies are actually relevant for human perception. In fact,it appears that the salient features for audio data tend to inhabit a low-dimensional subspace and only a subset of the frequencies present in perceptible music are non-zero.

Generation Experiments

We experimented with each of the music generation parameters under a variety of settings and qualitatively evaluated their performance. Unsurprisingly, it seems as though larger seed sequences tend to produce better results than shorter seed sequences, as the networks have a larger initial generation sequence and hence a larger context for their hidden layers. The number of iterations in which we perform sequence prediction also seems coupled with the choice of the initial sequence size. Empirically, generating a sequence that is roughly three times larger than the initial seed sequence tends to produce coherent music that does not result in generation loops (i.e. cases where the previous hidden state is very similar to the current hidden state and all proceeding hidden states become stuck in this pattern). It seems as though longer sequences eventually result in these self-perpetuating generation loops and the problem becomes quite common as the size of the seed sequence is reduced. We speculate that with larger training sequences we could likely mitigate this problem and it is likely that the 10 second training sequences we used are not sufficient for generating long songs spanning several minutes.

Visualizing the Network

In order to discover what our networks were really learning, we performed an experiment in which we predicted for every sequence in the dev set the corresponding sequence directly following it. We then computed the magnitude of each sequence and plotted the mean magnitudes over each 10 second example in the dev set to create a spectogram. The spectogram is shown in Figure 3. In the ideal scenario, our networks should be able to reproduce the ground truth mean spectrogram. As is evident from the figure, only a handful of frequency bands are active across time and correspond to frequencies that occur in natural music. While we did not explicitly sample frequencies within this band, it appears as though our models have learned to sample this band far more than the other frequency bands. Interestingly enough, the GRU network appears to have far greater activations outside of the band range of 4000 - 7000, which is likely why our GRU generated songs sound far noisier than our LSTM generated songs.

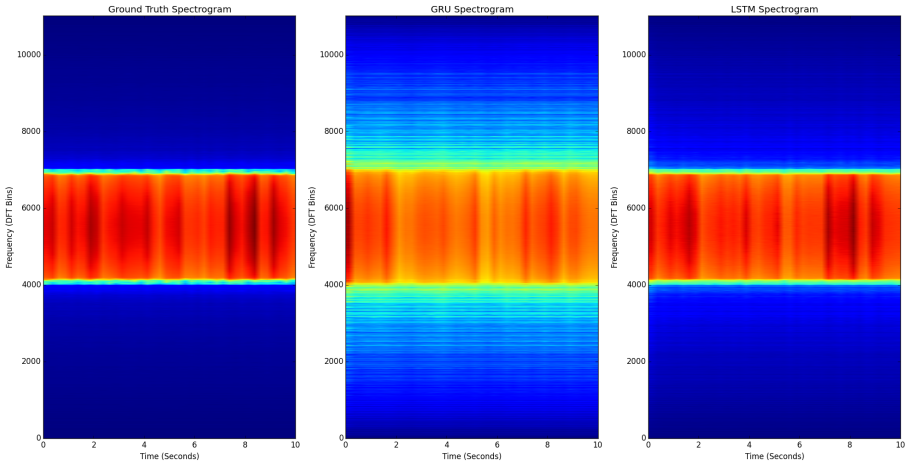

Figure 3: Comparison of the ground-truth spectrogram vs. the learned models' spectrograms.

Summary

The LSTM network consistently exhibited slightly lower training and validation loss than the GRU. However, on their generated outputs, the LSTM and the GRU network vastly differed. The generated outputs of the GRU were unsatisfactory and primarily consisted of white noise. On the other, the generated output of the LSTM network was musically plausible. Therefore, it appears that the added representation capabilities of the LSTM network resulted in a superior performance over the GRU network.

6 Conclusion

This project has demonstrated that algorithmic music generation with waveforms as input is possible with the use of recurrent neural networks, particularly the LSTM network. Interesting future directions include investigating the effect of adding layers of recurrent units and discovering the impact that additional layers have on performance. Similarly, it would be worthwhile to perform network training across genres with a substantially larger corpus. Unfortunately, due to the time and memory constraints of the Stanford Rye clusters, we were unable to pursue these directions fully. We are eager to experiment with more complex architectures and larger corpus sizes to see how well our preliminary results generalize and evaluate the effect of network depth on music generation performance.

References

[1] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. "On the properties of neural machine translation: encoder-decoder approaches". arXiv preprint arXiv:1409.1259, 2014. Presented at the Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8).

[2] F. Chollet. "Keras: Theano-based Deep Learning library". Code: `https://github.com/fchollet`. Documentation: `http://keras.io/`.

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling". arXiv preprint arXiv:1412.3555, 2014. Presented in NIPS 2014 Deep Learning and Representation Learning Workshop.

[4] D. Eck and J. Schmidhuber. "Learning the long-term structure of the blues". In J. Dorronsoro, editor, Artificial Neural Networks ICANN 2002 (Proceedings), pages 284289, Berlin, 2002. Springer. `http://link.springer.com/chapter/10.1007%2F3-540-46084-5_47`.

[5] S. Hochreiter and J. Schmidhuber. "Long short-term memory". *Neural Computation* **9** (1997): 1735-1780.