# Music Generation Using Recurrent Neural Networks

By KK Mokobi, Cade May, Gabriel Voorhis-Allen
Department of Computer Science, Stanford University

## Abstract

In this project, we trained Recurrent Neural Networks (RNNs) on musical data, with the goal of producing a model that could generate a song similar to its input. The input to and output from the network were MIDI musical files, which package all relevant song data (including note values, note timing, etc.). Data processing leveraged the Music21 toolkit to convert MIDI files into a sequence of integer-based categorical data (and vice versa). For training, the models were given 100-note sequences of a musical piece and tasked with predicting the 101st note; the loss function computed cross-entropy loss for this task. Our architecture search involved training a variety of RNN models, and we found success with models utilizing Long-Short Term Memory (LSTM) and Convolutional layers. The most successful model we trained, a 2-layer LSTM, 1-layer Conv1D hybrid, with dropout for regularization, achieved a training loss of .0097 and a testing accuracy of .7092. The models were trained exclusively on classical music data; however, we believe that these models can be trained to produce other genres of music through transfer learning.

## 1 Introduction

This project sought to produce a generative model which could produce artistic results after being trained on human-made creative art. Music is a very artistically rich domain, but can also be broken down into data (as it simply consists of sequences of pitch frequencies at varying rhythms and tempos). Thus, this project decided to attempt to use a model to generate classical music. Theoretically, such a model could serve many purposes: for example, generating new music similar in style to your favorite artist's music, or serving as the creative inspiration or foundation for a musical composition of one's own.

## 2 Related Work

For a discussion of the development of methods in the field of neural network music generation, and examples of machine-generated music, see [1] and [9]. For implementations of RNN music generation models, see [2],[3],[6] and [8]. [2], in particular, was relevant to our project, as the implementation therein served as the starting point for this project, and their model served as a baseline for our architecture search. While [2] provides a baseline, [3] is an example implementation that was able to generate music, but were limited in scope as a result of issues regarding encoding music files. [8] conducts a wide-ranging experiment on music generation methods, even testing simplistic methods such as N-grams. Finally, see [4] and [7] for an implementation of the distinct, but related task of generating sheet music using Generative Adversarial Networks (GANs) and Seq2Seq models.

## 3 Data Processing

### 3.1 Datasets

For our dataset, we collected classical music stored in MIDI files. MIDI files effectively package the necessary temporal musical data, including note values, timing and song titles, that we need to train a model to generate music. We chose to train and test on classical music, as there is a wealth of data for this genre easily sourced on online databases.

We included classical piano music from composers such as Beethoven, Waldstein and Chopin. We built the training, dev, and test sets so that they contained songs with consistent time and key signatures, allowing the model to focus on rhythm and pitch sequencing without encountering conflicting keys or beats per measure. The dataset exhibited a 10/3/3 split in terms of the number of songs for training, development, and testing data. For each training set, these 10 songs translated into approximately 10,000-30,000 sequences of musical data of length 100ms each.

### 3.2 Data Pre-processing

To parse our MIDI files into distinguishable note values we used the toolkit, Music21. The (convertor.parse) function enabled us to parse MIDI files into an array of objects -- either notes or chords. These objects contain information on pitch type and note duration, which we used to generate a list of strings that categorize observed notes. We converted this data into a number array using an associative array created from the set of total notes observed. Models perform better on integer categorical data in comparison to strings or larger note types.

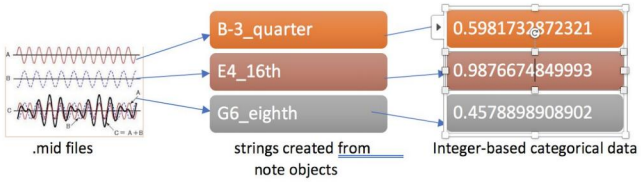


Figure 1: Illustration of data pre-processing

### 3.3 Method

Generation of music requires a model to understand temporal correlations from sequential data. LSTM- based RNNs are a common model used with sequential data, and these models perform better on integer-based categorical data. To generate music, we attempted to predict a note using the sequence of notes before it. Sequence length heavily affected our models performance, but as a hyperparameter we commonly set it to 100. We fit our models using this data, and tested multiple different model sizes. We evaluated models using categorical cross entropy during training to optimize our outputs.

Once trained, a randomized input is used on our model to predict a new song, which we convert back to a MIDI file using similar functions as described in the previous pre-processing steps.

## 4 Models

### 4.1 Architecture Search

We evaluated a wide range of architectures. Due to the temporal nature of audio data, we spent the majority our time working with RNN-based models. A lot of the experimentation went into identifying other structures to wrap around these layers in order to help them extract meaning from the data.

The early phase of our architecture search was a highly empirical process. We simultaneously ran 5 different Amazon ec2 instances so that we could quickly iterate through different model types. After finding that models tended to begin showing their true colors after around 200 epochs of training, we settled on this as the point at which would evaluate and compare models. We then proceeded to select models to continue training and working with based on their performance after 200 epochs.

**Baseline Model**

The baseline architecture was a simple recurrent neural network. It consisted of four LSTM layers in sequence. While this model was reasonable in its ability to fit the training data, it exhibited symptoms of high variance. We attribute this to the simplicity of the approach. We believe that music is too nuanced and dynamic to be captured by such a conservative architecture. Our modelling decisions reflected this belief; we sought to experiment with a wide variety of complex, hybridized models. We ended up finding our most promising model at the intersection of convolutional and recurrent architectures. We now outline some of the techniques we experimented with:

**Convolution:** We postulated that convolving a filter over each input sequence could offer a smoothing

effect, allowing our RNN model to more easily derive patterns from the data. In any sequence extracted from a song, the notes follow a specific pattern. The hope was that passing filters over local sections of each note sequence could help to regularize these rigid, numerical patterns. Ideally, this allows the model to take advantage of the parameter-sharing property of convolution through the application of global patterns to localized sequences of notes.

**Long-Short Term Memory (LSTM):** Given the importance of long-term patterns in musical data, we needed a method to incorporate "memory" into our model. While we experimented with Gated Recurrent Units (GRUs), we found Long-Short-Term Memory (LSTM)-based models to yield superior results. As a result, most of our models utilized one to three LSTM layers.

**Time-Distributed and Double Convolution:** We found that in isolation, time-distributed-dense layers struggled from overfitting. However, in combination with other layers in deeper models, these time-distributed layers performed well.

**Regularization:** To combat overfitting, we experimented with a variety of regularization techniques, including dropout and batch normalization. We found that dropout with high keep-probability produced the best results; BatchNorm and aggressive dropout were not as successful.

### 4.2 Hyperparameter Tuning

The main two types hyperparameters that we found utility in training were the number of hidden units and the number of layers. Finding the optimal number of units for recurrent layers presents significant trade-offs. On one hand, decreasing the number of hidden units means the network derives a less complex function. Most of our models used 256 hidden units per layer, and decreasing this number to 128, without compensating by changing any other parameters, yielded inferior musical outputs.

On the other hand, the reduced computational demands of small hidden units allowed us to experiment with deeper networks. We found that halving a model's hidden unit count while also doubling its number of layers results in a new model with comparable performance, but lower computational requirements. To quantify this disparity in energy demands, consider a 2-layer LSTM model with 256 hidden units in each layer; we built a model like this and it has over 5.4 million parameters. Now consider a 4 layer LSTM with 128 hidden units in each layer; our model of this construction has just 2.5 million parameters. With respect to complexity and computational limitations, it certainly proved worthwhile taking hidden unit reduction into consideration.

## 5 Results, Metrics, Discussions

### 5.1 Table of Results

| **Build:** | **Baseline**: 4 LSTM-256, Dropout | **Model 1:** Conv1D-256, 2 LSTM-256, Dropout | **Model 2:** 1 LSTM-256, Dropout | **Model 3:** Conv1D-128, 2 GRU-256, Dropout | **Model 4:** Conv1D-256, 3 LSTM 128, Dropout | **Model 5:** Conv1D, 2 LSTM-256, BatchNorm | **Model 6:** Conv1D, TimeDense, LSTM, Dropout | **Model 7:** Conv1D, TimeDense, LSTM, Conv1D, LSTM |
|---|---|---|---|---|---|---|---|---|
| **Train Loss:** | 0.0139 | 0.0097 | 0.0139 | 0.0078 | 0.0035 | .0186 | 0.0028 | 0.0017 |
| **Test Loss:** | 0.8535 | 0.7092 | 0.7834 | 0.8375 | 0.7902 | 0.7539 | 0.8724 | 0.6984 |

Figure 2: Training and testing loss of selected models

$$CCE = -\frac{1}{N}\sum_{i=0}^{N}\sum_{j=0}^{J} y_j \cdot log(\hat{y}_j) + (1 - y_j) \cdot log(1 - \hat{y}_j)$$

Figure 3: Cross-Entropy Loss Function

The table above (fig. 2) exhibits our quantitative approach to evaluating a small handful of the models we worked with. Since our model trained on categorical note values, we chose to evaluate using categorical cross entropy loss (fig. 3). The values shown were taken after 200 epochs of training.

### 5.2 Discussion of Results

Quantitatively, all of our models displayed symptoms of high variance. However, we found that there was little correlation between the cross-entropy loss value and the qualitative quality of the musical outputs. The quality of the model's output also seemed to consistently vary within a given song; certain sections offered spurts of promising melodies, while other sections sounded dissonant and chaotic. We will now discuss results from a few relevant models:

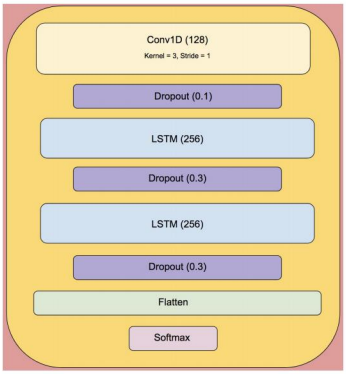


Figure 4: An illustration of Model 1.

**Model 1: Conv1D-256, 2 LSTM-256 with Dropout**

Our highest quality musical output came as a result of attaching convolution layers to LSTM-based RNN. We believe that convoluting a size-3 filter over the input sequences enhanced the model's ability to understand musical patterns. The early success of this model inspired us to invest more training time in it. We trained it to well over 2000 epochs and, it became the only model that consistently generated output sequences that sounded like they were in the same musical key.

**Model 2: One-Layer LSTM**

We experimented with a shallow RNN in order to help us forge a better understanding of the value of hidden layers. The outputs of this network were predictable at best. At worst, like many other models we tried, the shallow RNN produced erratic and inharmonious sounds.

**Model 4: Conv1D-256, 3 LSTM-128, Dropout**

This model was an attempt at evaluating the utility lower hidden unit counts. We increased the number of hidden layers in order to allow the deeper model to retain some level of complexity, and we halved the number of hidden units in order to allow it to train faster. Qualitatively, this model performed in a comparable manner to its larger counterparts.

**Model 7: Conv1D, TimeDense, LSTM, Conv1D, LSTM**
More complicated models seem to yield more dynamic and nuanced musical outputs. The outputs of this model immediately stood out. While some other models seemed to generate predictable sequences, even as bad as continuously playing the same chord, this model generated an incredibly diverse range of notes and patterns, even over a very short period of time.

## 6 Future Work

We had lofty goals to start this project: we envisioned a general-purpose music generator which, given a MIDI file of music, would produce similar-sounding music. We were able to accomplish our core task; the different models we trained generate music of varying degrees of quality, and our best model (a two-layer LSTM network with a Conv1D layer and dropout) produced adequate-sounding music after being trained for over 2,000 epochs. However, our project could be improved by loosening the simplifying restrictions that we put in place along the way.

We plan to improve our networks to be robust to larger and more complex datasets: music from a wider array of genres, with more instruments, and with varied time and key signatures. We believe that the model we've trained will be largely transferrable to accomplishing this more complex task, although we will need to perform many more iterations than the 2,240 we achieved this time. We do plan further changes to the model itself as well, including incorporating dimensionality reduction to produce less complex, but better-sounding output and or adjustment of sequence length to better understand models predictive power.

## 7 Github Repository

https://github.com/cademay/cs230-music-generation

## 8 References

[1] McDonald, Kyle. "Neural Nets for Generating Music." Artists and Machine Learning, Medium, 25 Aug. 2017
[2] D. Gallegos and S. Metzger. "LSTiestoM: Generating Classical Music." CS230: Deep Learning, Winter 2018.
[3] Skúli, Sigurður. "How to Generate Music Using a LSTM Neural Network in Keras." Towards Data Science, Medium, 7 Dec. 2017.
[4] N. Agarwala, Y. Inoue, and A. Sly. "Music Composition using Recurrent Neural Networks." CS 224n: Natural Language Processing with Deep Learning, Spring 2017.
[5] . B. Sturm, F. Santos, and O. Ben-Tal. "Music transcription modelling and composition using deep learning." Cornell University Library, 29 April 2016.
[6] S. Pai and I. Goodman. "CS 224D Final Project: DeepRock." CS 224d: Deep Learning for Natural Language Processing.
[7] K. Huang, Q. Jung, and J. Lu. "Algorithmic Music Composition using Recurrent Neural Networking." Stanford University.
[8] Wang, Jason. "Deep Learning in Music." Stanford University, 16 Dec. 2016.
[9] Brinkkemper, Frank. "Analyzing Six Deep Learning Tools For Music Generation." The Asimov Institute, 5 Oct. 2016.