A Harmonious Approach to Algorithmic Composition

Nicholas Graziano
ngraz@stanford.edu

Abstract

A predictive text based approach was implemented to suggest a successive chord to a composer given a preceding chord progression. While it proved relatively easy to implement it was limited by the mapping of each chord to a single feature.

INTRODUCTION

It is hard to remember a world before predictive text. Smartphones now offer complete sentences as prospective responses to text messages and emails. It is vanishingly rare to interact with a user interface without it trying to figure out what you are going to input before you do. If machine learning can be used to offer the next entry in a language application, could a similar function not be achieved in a musical setting?

A chord is a collection of notes characterized by the intervals between its component pitches. In classical music theory, certain chord forms are considered to "want" to resolve to others. For example, the dominant seventh chord feels as if it should progress to its relative fourth, to an ear accustomed to western musical conventions. However, the majority of chords do not imply a particular successor in their structure alone.

Language models have used long short-term memory recurrent neural networks to map input sequences to vectors of fixed dimensions. A similar approach could be applied to this musical application with different learned weights applying to preceding chords of increasing distance.

RELATED WORK

Machine learning based music composition is not a novel application but it has generally functioned as a black box which returns a song or melody actualized as a waveform or encoded in midi file. [1] This is an incredible technological feat but not necessarily useful as a tool to assist a human composer.

In order to create a more user friendly interface this was approached more similarly to a text prediction application. Long Short-Term Memory (LSTM) is a type of neural network that has become popular for sequence to sequence mapping in language processing. [2] One reason it is popular for this application is that is capable of taking inputs of variable length and returning outputs of different lengths. Another strength is that the recurrent nature of LSTMs allows it to learn on long sequences of data with temporal dependencies.

DATASET AND FEATURES

There are a number of existing popular music datasets in various formats as outlined by Wang. et al. [3]. Many of these contain melody information for the corpus but far less include an easy to process sequence of chords. In their paper Wang et al. propose a new dataset of 909 pop songs titled POP909. It includes time sequence chord data for each song presented in a natural form for a human composer, e.g. G:maj for G major or E:min7 for an E minor seventh chord. Besides the use of the colon as a delimiter these are roughly identical to the form a musician would be used to seeing.

A first pass through the corpus was used to index the unique chords. The "N" used for a rest or "none" was removed and the remainder were stored. As an experiment chords with less than 100 instances in the corpus were dropped, however this was omitted from the final model as it didn't appear to affect the accuracy and would limit the acceptable inputs. A distribution of the five most common chord types in the corpus for each base note is provided in Figure 1.

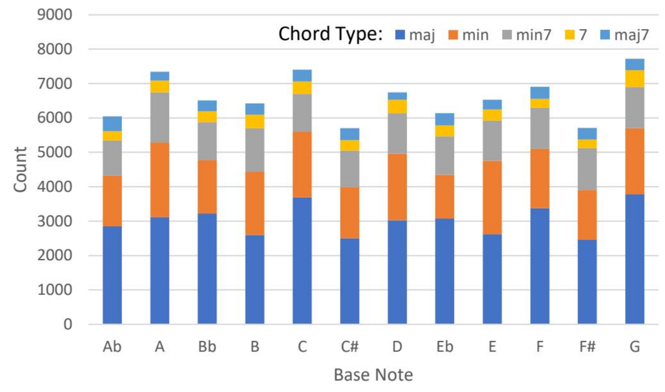

Figure 1. Chord Distribution for Five Most Common Chord Types

A second pass was made through the dataset. Each song was stored as a string of integers corresponding to the index where each constituent chord was stored in the dictionary. A sample of the first five chords of the first song are given below both as represented in POP909 and stored in the data set. The distribution of the number of chords in each song is given in Figure 2.

```
B:maj7 C#:maj Bb:min7 Eb:min B:maj7

186, 390, 287, 653, 186
```

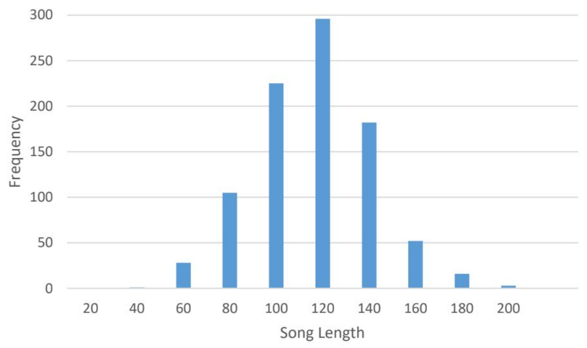

Figure 2. Song length distribution

The entire corpus was stored as a PyTorch dataset. When training and validating the model a random five chord sequence was sampled from each song for use where the first four considered an input and the last four used as an output. It was determined that the influence of any chord more than four prior was of diminishing value. For a manual inquiry, only the last four chords were fed in to the model. For an input of less than four chords, the earliest chord was appended to the front of the sequence until it reached a length of four.

METHODS

Recurrent neural networks (RNNs) function by maintaining a hidden state which acts as a sort of temporal memory allowing the interpretation of a sequence rather than a single input. Long short-term memory describes a novel approach to recurrent neural networks put forth by Hochreiter et al. in 1997. [4] It is able to store information over a longer time delay by instituting lags in the feedback loop and gates which can be trained to determine the relevant information to feed forward.

Deep learning is the process in which multi-layered neural networks update. It consists of a forward and backward step. In the forward step the model parameters are updated per the new inputs. On the back step the predicted outputs are compared to the actual outputs via a loss function and updated per an optimizer.

PyTorch is a Python package optimized for deep learning applications with a built in LSTM architecture. The forward step of its implementation consists of updating the input, forget and output gate values using the sigmoid function for activation and the cell gate using hyperbolic tangent function. An activation function is a function that maps any input to either or a 0 or 1. Using these weights it then updates the hidden state and cell state.

Adam is a popular optimizer for neural networks built into PyTorch. It functions efficiently only needing to store first order gradients and using optimized time steps to update its estimates parameters. [6] It also includes a dropout feature by which random data from the input vector will be zeroed out in order to regularize the model and avoid neuron collapse. [7]

A model was adapted for chord based input from a standard predictive text setup. [8][9]

Sequences of five integers representing chords were sampled from random positions within each song in the dataset. The first four were fed in to the model as an input. The cross entropy loss was then calculated between the output and the last four integers. A backward step was taken and optimized using Adam.

The model was initially setup using 2 layers, 32 for both the hidden and embedding layer dimensions, and a dropout rate of 0.2. The default Adam learning rate of 0.001 was utilized. More detail will be given on how these were optimized in the discussion section.

DISCUSSION

Model convergence was measured using PyTorch's Cross-Entropy Loss function. This function is chosen as it expects an integer input similar to the indexing used for storing the chords rather than an encoded vector. [10] The equation for cross entropy loss is given by the equation below.

$$loss(x, class) = -x[class] + \log\left(\sum_j \exp x[j]\right)$$

While the moving average of the cross-entropy loss did seem to converge to some degree, the value at each step was extremely erratic, varying at a magnitude greater than that of the average convergence as illustrated in Figure 3.

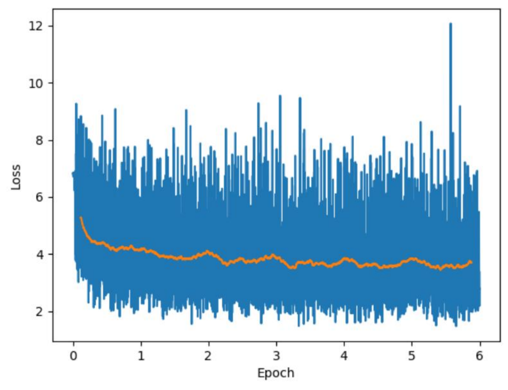

Figure 3. Cross-Entropy Loss

Experiments were conducted, varying the learning rate and introducing a weight decay used in the optimization step. While it was possible to manipulate these to produce a less erratic plot it was clear that it was as at the cost of attenuating the weights to such a degree that practically no learning at all occurred before the influence went to zero. It is possible that a larger batch size could have reduced the loss at each step but unfortunately the way the model was formulated the time sequence of the chords occupy the dimension where PyTorch's dataloader attempts to expand the number of samples. Changing this proved nontrivial and impractical due to time constraints.

While the loss values were erratic they did not appear to be divergent so the hyperparameters were instead tuned based on model accuracy. The accuracy was measured by feeding the previously partitioned test data through the predict module of the model. As in the training portion the first 4 chords of the sequence were treated as input, the last integer of the output was then compared to the index of the fifth chord in the sample. An initial sensitivity was run for learning rates of descending orders of ten. Once the optimal order of magnitude was determined a finer resolution test was run to select the ultimate value of 1.097E-03 as indicated in the figures below.

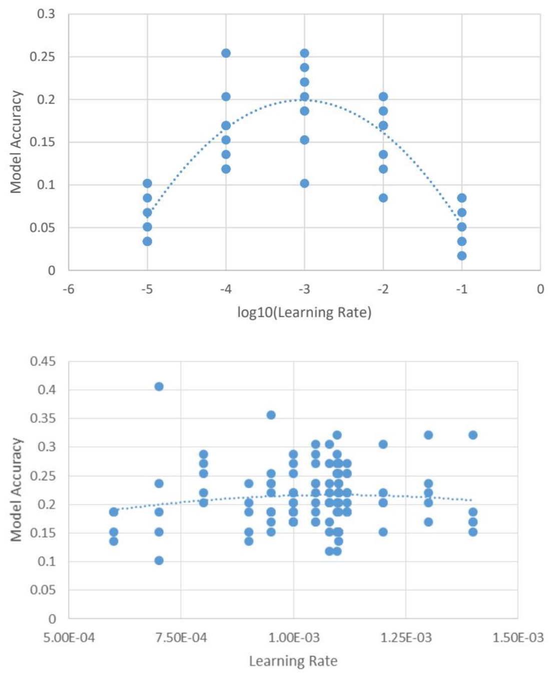

Figure 4. Learning Rate Determination Trials

As mentioned previously a weight decay function was also examined. The weight decay parameter of PyTorch's Adam optimizer is somewhat similar to a L2 normalization term although not implemented identically as examined by Loshchilov et al. [11]. As the model did not appear to actually diverge the weight decay only acted to reduce learning and thus accuracy and was ultimately omitted from the model formulation.

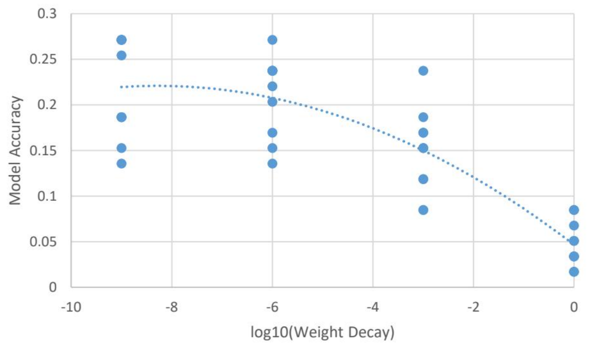

Figure 5. Weight Decay Optimization

Once the optimizer parameters were set a similar battery was run on the model architecture parameters. Initial trials were run varying the number of layers and the dimensions of both the hidden and embedding layers of the neural network. Ultimately two layers was selected as it appeared that model accuracy only dropped off with the inclusion of additional layers as indicated in Figure 6.

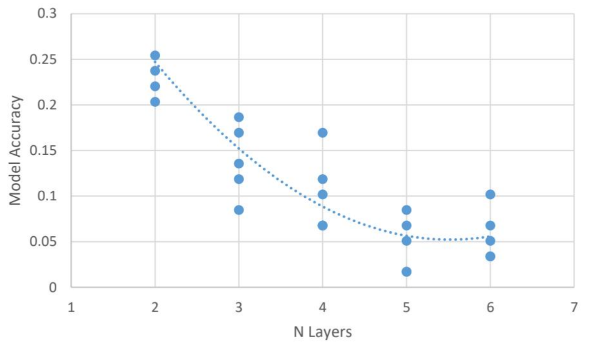

Figure 6. Layer Count Optimization

Once the number of layers was solidified another test matrix was executed on the network dimensions. Both the ratio between the layer dimensions and the absolute dimensions were considered. An optimal ratio of 1:2 for hidden to embedding was determined. Absolute sizes of 512 and 256 respectively were selected due to the observed diminishing returns.

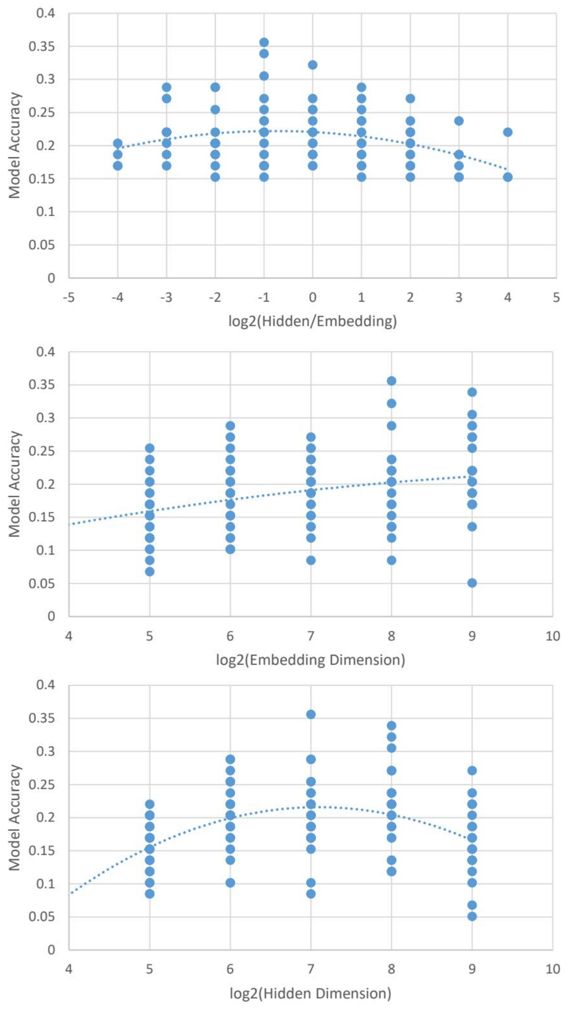

Figure 7. Dimension Optimization

Next an optimal drop rate of 9.336E-02 was determined.

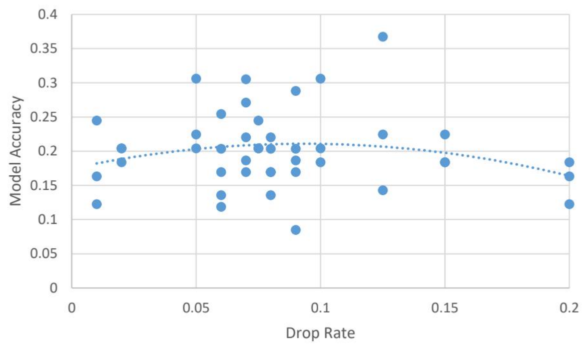

Figure 8. Drop Rate Optimization

Although set in the beginning and tweaked throughout model formulation a final sensitivity experiment was run on the number of samples used to train the model and the amount of epochs once the rest of the parameters were solidified. To accomplish this the full dataset was partitioned for the selected number of training samples and the rest were used for testing. Given that the dataset was only one sample selected from each of the songs in the corpus it should be noted that the test size of 59 corresponding to a training size of 850 is considerably less than that of the 709 remaining for a training size of 200. Nonetheless the model accuracy appeared to plateau at around 5000 total iterations $(n_{iter} = N_{epochs} \times n_{train})$ regardless of the combination of training size or epoch count.

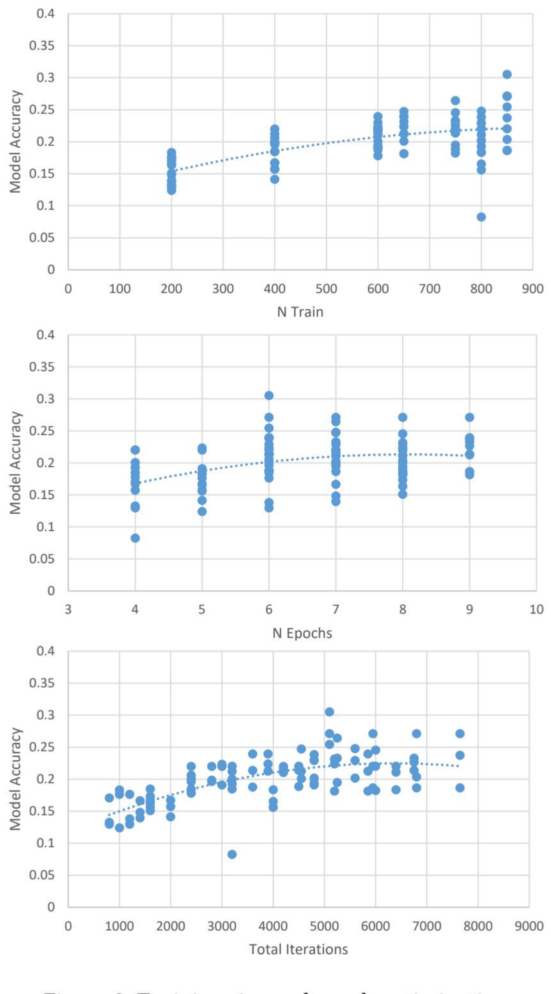

Figure 9. Training size and epoch optimization

Table 1. Final parameter values

Parameter	Value
$r_{learning}$	0.001097
n_{train}	850
N_{epoch}	6
n_{layers}	2
$d_{embedding}$	512
d_{hidden}	256
$r_{dropout}$	0.09336

Given that only 5 chord long sequences were taken from each song for each training/validation session it was possible to specify different random seeds and lean on the >90,000 possible overlapping 5 chord long sequences in the corpus throughout development. A new seed was chosen to run a final test on all 909 songs.

The amount of chords guessed correctly by the first suggestion was 16.2%. The percentages guessed within the first two and three suggestions respectively were 22.1% and 25.9%. For a user test of "E:min C:maj7 E:min9 C:maj A:min" corresponding to the first 5 chords of Pink Floyd's *"Welcome to the Machine"* the model suggested the following chord of D minor which is unfortunately an inferior choice to the E minor actually used in the song.

CONCLUSION

A predictive text based approach to musical composition offers a relatively simple to implement and interpret model. However, this naïve setup fails to leverage so much of the underlying mathematical structure of music. Modeling each chord instead as its constituent notes, either as a series of integers or a multi-hot vector could enable better mapping to the actual features that make up a chord rather than a single arbitrary index. Similarly it would be interesting to explore how the relationship between relative chords could be generalized as the movement from C major to A minor is equivalent to the movement from G major to E minor. A better use of the data in this corpus could have been to use all of the 5 chord sequences in the data rather than one per song and to incorporate the key each song was in as an additional feature.

REFERENCES

[1] Briot, Jean-Pierre, Gaëtan Hadjeres, and François-David Pachet. "Deep learning techniques for music generation--a survey." arXiv preprint arXiv:1709.01620 (2017).

[2] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." arXiv preprint arXiv:1409.3215 (2014).

[3] Wang, Ziyu, Ke Chen, Junyan Jiang, Yiyi Zhang, Maoran Xu, Shuqi Dai, Xianbin Gu, and Gus Xia. "Pop909: A pop-song dataset for music arrangement generation." *arXiv preprint arXiv:2008.07142* (2020).

[4] Sepp Hochreiter, Jürgen Schmidhuber; Long Short-Term Memory. Neural Comput 1997; 9 (8): 1735–1780. doi: https://doi.org/10.1162/neco.1997.9.8.1735

[5] "LSTM." LSTM - PyTorch 1.8.1 documentation. Accessed June 3, 2021. https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html.

[6] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[7] "Dropout." Dropout - PyTorch 1.8.1 documentation. Accessed June 3, 2021. https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html.

[8] Bitvinskas, Domas. "PyTorch LSTM: Text Generation Tutorial." KDnuggets, July 2020. https://www.kdnuggets.com/2020/07/pytorch-lstm-text-generation-tutorial.html.

[9] Muñoz, Eduardo. "Intro to RNN: Character-Level Text Generation With PyTorch." Medium. Better Programming, February 11, 2021. https://betterprogramming.pub/intro-to-rnn-character-level-text-generation-with-pytorch-db02d7e18d89.

[10] "CrossEntropyLoss." CrossEntropyLoss - PyTorch 1.8.1 documentation. Accessed June 2, 2021. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html.

[11] Loshchilov, Ilya, and Frank Hutter. "Decoupled weight decay regularization." arXiv preprint arXiv:1711.05101 (2017).