Deep Learning Music Generation

Kinbert Chou
Stanford University
Stanford, CA 94305
`klchou@stanford.edu`

Ryan Peng
Stanford University
Stanford, CA 94305
`pengryan@stanford.edu`

Abstract

We are interested in using deep learning models to generate new music. Using the Maestro Dataset, we will use an LSTM architecture that inputs tokenized Midi files and outputs predictions for note. Our accuracy will be measured by taking predicted noted and comparing those to ground truths. Using A.I. for music is a relatively new area of study, and this project provides an investigation into creating an effect model for the music industry.

1 Introduction

Music is deeply embedded in our everyday lives from listening to the radio to YouTube music videos. With everyone having a distinct music taste, the area of music is only expanding. Everyday, new songs are being created and the art of music holds the interests of many all across the world from different countries and cultures. We are interested in exploring the intersection between music and A.I. In recent years, deep learning has reached a level of generating words such as natural language processing (NLP), however, there is less research done on generating music. While there have been projects aimed at generating new music, this project poses a front on whether music generation using an existing model language architecture is achievable. Our project tackles the category of music generation using classical music. We plan on modeling musical data similarly to human language in our projects.

2 Related Work

There are existing implementations built using a recurrent neural network architecture and a few that explore the use of long short-term memory (LSTM) architectures. In Yu's article (1), recurrent neural networks (RNNs) are inefficient at generating music. The research shows that vanilla neural networks are bad at sequential data generation. Combined with Briot et al.'s study "Deep Learning Techniques for Music Generation" (2), RNNs are known to have problems such as vanishing and exploding gradients when the network is too deep. This is solved by using LSTMs which basically create shortcuts in the network We used a LSTM model because its cell state can carry information about longer-term structures in music as opposed to a gated recurrent unit (GRU). To improve the algorithm, changing the hyperparameters and choosing our own set of hidden layers in accordance with data piano music may prove to be the best option. An advanced component of this project we implemented was feeding in tokenized MIDI data into a LSTM model similar to what is typically used to generate text sequences, which is derived from Skúli's data preparation for tokenizing MIDI files (3). Spezzatti's study shows tips for encoding, tokenizing, and sequencing lyrics as well as an analysis of different state-of-the-art music generation examples including Magenta, MuseGAN, Wavenet, and MuseNet (4). Each using different encoding methods and architectures, but also states shortcomings with each model (notes not in key, excessive repeating of notes, etc.) Another component to consider is the evaluation metrics for a model. From Jung's analysis, many models use a variety of evaluation techniques all of which emphasize certain qualities in a song like rhythm, uniqueness, and aesthetic.

While most of these metrics are more subjective, Jung proposed an evaluation model that considers the structure of the song (intro, verse, bridge, chorus, etc.) using a self-similarity matrix (5).

3 Dataset and Features

We used the MAESTRO dataset (6) for our project which comes from a leading project in the area of processing, analyzing, and creating music using artificial intelligence. The dataset consists of over 200 hours of piano music. The dataset is well defined and cleaned: the dataset includes MIDI files from over ten years of International Piano-e-Competition. The genre for the music files is mostly classical with composers from the 17th to early 20th century. The metadata has the following fields for every MIDI/WAV pair: canonical composer, canonical title, split, year, MIDI filename, audio filename, and duration. We follow the train, validation, and test splits defined by the metadata .csv file.

3.1 Preprocessing

All of the data was put into three categories (train, validate, and test). The MIDI files we are working with will be read using Music21. The data contains the notes object type and this contains information about the pitch, octave, and offset of the note. For the base model there are 967 samples for the train set, 137 samples for the validation set, and 178 samples for the test set. We start by loading each file into a Music21 stream object using the converter.parse(file) function. Then, we get a list of all the notes in the MIDI file. Next, we append the pitch of every note object using its string representation since the most significant parts of the note can be recreated using the string notation of the pitch. We tokenize those string outputs to feed it into the network. For each example, we use a sequence of the 100 preceding notes in order to predict the next note. We continue this "sliding window" process until we have seen all notes in the file. Our model is fed these inputs of "window," note pairs. These encodings allows us to easily decode the output generated by the network into the correct notes. We write this processed data to our data folder and load them at time of training. This preprocessing workflow is heavily adapted from the work of Sigurður Skúli (3).

4 Approach

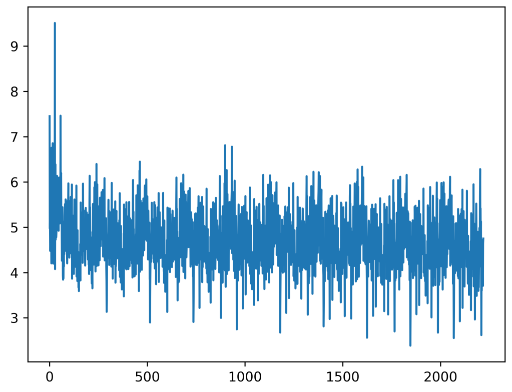
The current general architecture of our model consists of using a single long short-term memory (LSTM) layer that takes in a sequence of note tokens generated from our data and outputs the predicted token given our vocabulary. The model takes in the input and passes through three LSTM layers. The first layer turns the note indexes into hidden state encodings. The second layer then refines the hidden state outputs from the first layer. The third layer takes in the refined hidden state outputs, and we pass the third layer's last hidden state to a series of fully-connected layers to make predictions. This is followed by a layer of batch norm and two linear layers to produce a vector the same dimensions as our vocabulary size. There is also a layer of batch norm and ReLU activation between the linear layers. The last fully connected layer maps the output of the previous linear layer to a vector the length of our vocabulary. We pass this final output to a softmax function, and this final softmax output represents what our prediction for the note is. LSTMs are good at processing sequential data which music is an example of. The notes leading up to a note in question will inform the model of what the next note will be, as the previous notes hold crucial information. For our loss function we are using the cross-entropy loss given that we are predicting the next token from our input. We also use the stochastic gradient descent optimizer in our training. We closely follow the structure stated in: https://github.com/Skuldur/Classical-Piano-Composer, but recreate the architecture using PyTorch.

5 Training and Evaluation

5.1 Training

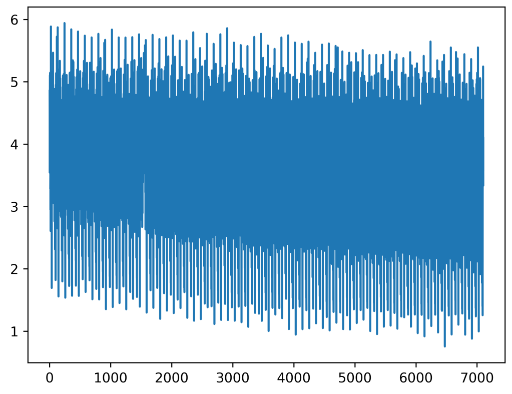
We trained our model on 2839786 examples of (length-100 sequences, next note) pairs generated from roughly 1000 midi files. Training for the model on a p3.2 large instance on AWS took roughly

36 hours. The following plot diagrams the training loss values over the first 10 epochs.



Training loss for first 10 epochs

From the progression of loss over iterations, we can see that the loss decreases, albeit minimally, for each epoch, and roughly achieves the minimum loss at the same point for each epoch. However, for each epoch, loss can vary widely. We believe this can be explained by the structure of the data: notes, chords, key signatures, and other musical features can vary widely from song to song. Thus, over one epoch, the model will encounter notes that occur very often, and can update its weights easily, and notes that occur infrequently in the dataset. Because we are not using embedding representations as do similar tasks such as next-word prediction, the model must first learn its own internal representation of each note/chord token. Thus, some songs can be inherently easier because they are in a common key signature, and some can be more difficult.



Training loss for our model with batch size 128, epoch 20-100

For the remaining epochs, the model shows consistent decrease in average loss value over time, but still oscillates in a large range between easy and hard songs. The range for which the loss oscillates is very high as compared to a model with a lower batch size.

5.2 Hyperparameter Tuning

We trained a few times using different parameters to analyze loss and accuracy. We trained using a batch size of 32, 64, and 128. We also trained the model for 30, 50 and 100 epochs to compare. The model with the best results contained a batch size of 128 with 100 epochs. This model achieved lower loss and higher accuracy compared to other models.

5.3 Results and Evaluation

Table 1: Accuracy & Loss for Different Models

Model	Accuracy	Loss
128/100 Train	0.0460	1.62
128/100 Validation	0.0152	
128/100 Test	0.0146	
64/85 Train	0.0108	4.27
64/85 Validation	0.0134	
64/85 Test	0.0124	
32/30 Train	0.0108	4.81
32/30 Validation	0.0134	
32/30 Test	0.0124	

5.4 Error Analysis

We observe that while the model does poorly in training accuracy, it performs even worse in dev and test sets. There are several possible reasons for this: Analysis of our model's predictions show that for Validation and Test sets, it is mostly predicting the notes G or D, at different octaves. These two are very common notes in many keys in music. During training, the model could have learned to predict these notes frequently to reduce loss. One proposed solution to address this is to train the model for more epochs, allowing it to learn better representations of input sequences. Additionally, the dev and test sets can possibly come from a different distribution than the training set. Pieces in these sets can have new key signatures, or new and unseen chords. One suggested improvement to address this is to model our input data similar to character embeddings used in NLP deep learning models. This will allow the model to generalize its learning to unseen chords.

Another possible error in the approach is that the vocabulary is simply too complex to learn. To test this, we train on a significantly reduced dataset, roughly 12% of the actual, and we expect the model to overfit to this set. However, do to either a low number of epochs (30) or an insufficiently complex model, the performance on this smaller dataset is also poor.

6 Conclusion

With our training model, the loss does generally decrease with each subsequent epoch. Since the training is done with a variety of songs, harder more complex songs have a higher loss to learning. This is mainly seen in the increase and decrease in loss over time within a single training iteration.

Our results show that a batch size of 128 with training under 100 epochs produces the best accuracy. The accuracy for the train set produces a higher accuracy than the test and validation as expected, but is still significantly underfitting. Accuracy is still low as the model requires more epochs, additional tuning, and possibly a deeper network. For further work, finding a dataset with similar songs may be a good start to improving the baseline for this model. A lot of fluctuations for the loss during training attributed to different songs containing notes that are harder to interpret than others.

Given the data gathered during this study, the area of music generation requires more training and model structure. From other studies, music generation using a LSTM model needs many epochs to accurately predict next generative notes. This is coupled with a complex multi-layer architecture in most models for maximum accuracy. In the future, this model can be improved by adding another hidden layer and a method for processing unique notes in the validation and test set data. Ultimately, the area of music generation is quite large and once a model is established, there can be more research done for different music industries (pop, rock, EDM, etc.) and while piano music generation is a start, building off of these models will be new entry into a creative way to make music.

7 Github Repo

https://github.com/kl-chou/CS-230-Project

Our work can be found in the LSTMModel folder in the above GitHub repository. get_notes() and prepare_sequences() is adapted from the Classical-Piano-Composer repository, the work of Sigurður Skúli.

References

1. https://towardsdatascience.com/neural-networks-for-music-generation-97c983b50204
2. https://arxiv.org/pdf/1709.01620.pdf
3. https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5
4. https://towardsdatascience.com/neural-networks-for-music-generation-97c983b50204
5. https://towardsdatascience.com/making-music-when-simple-probabilities-outperform-deep-learning-75f4ee1b8e69
6. https://magenta.tensorflow.org/datasets/maestro