CLASSICAL MUSIC GENERATOR

Zhuoyang Lyu
Yixiang Sun
Zixuan Guo

DATA: MusicNet

We found a publicly available dataset on kaggle: MusicNet, a curated collection of labeled classical music. “MusicNet is a collection of 330 freely-licensed classical music recordings, together with over 1 million annotated labels indicating the precise time of each note in every recording, the instrument that plays each note, and the note‘s position in the metrical structure of the composition. The labels are acquired from musical scores aligned to recordings by dynamic time warping. The labels are verified by trained musicians; a labeling error rate of 4% has been estimated. The MusicNet labels are offered to the machine learning and music communities as a resource for training models and a common benchmark for comparing results.” We plan to only use the annotated note as our data, not the raw audio files. We will need to embed the notes to an embedding space not only of its pitch but its length and possibly its instrument. Now, we create a simplistic dataset that reduced data variation to minimum, that only contains notes' degree after modulating to C Major key and note duration. All the other information discarded. This data, should be easier for our model to learn the relationship between notes.

METHODOLOGY: RNN

Because of the assemblance between music data and natural language data, we decided that Recurrent Neural Network would be a potential solution, so we used RNN to approach this problem. We experimented and trained multiple LSTM with different setup on different dataset and compared their output. We used not only the midi-track data but also the note feature embedding dataset, simplified and not simplified. The midi track data can directly output midi
data that is easy to evaluate and contains much more information. On the other hand, the note feature embedding is easier to interpret, but is harder to train and evaluate. The information like composer, instruments that don‘t necessarily affect the music itself as much as the note are weighted as much as essential features like pitch and timing. Those information made the dataset much more variant and complex. The result, being a list of generated notes with features, cannot be played and listened to directly. We can only evaluate it by quantity, model loss, but not its tonal quality. We have to manually transpose the music on a sheet to hear it. Therefore, we prepared the simplistic dataset and discarded all the unnecessary information and saved only the essential. The output will be a sequence of notes with only pitch and time .Transposing that melody is much easier and straight-forward.

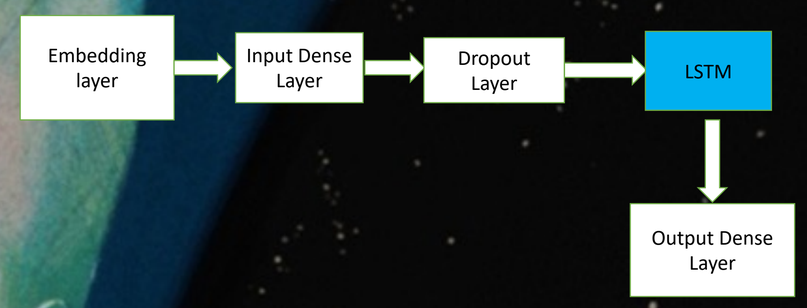
Models and Results

Basic LSTM Model (Baseline, Dummy without creativity)

We first train a basic LSTM model using the midi input as our base-line model.
The model is made of one LSTM layer with 512 units, taking in input directly and pass the output through a dropout layer with rate 0.5, then a final dense layer preparing us for the final compatible result. We trained the model with the most commonly used state of the art hyperparameters, mean absolute error as loss and Adam optimizer, and a batch size of 256, trained for 4 epochs. The loss is undesirably large, and the music generated is undoubtedly not what we wanted. It is always chromatically ascending/descending, so in other words, lacks the agreeable succession or arrangement that defines satisfying music. What we are looking for is the musically satisfying sequences of collective notes. To upgrade our model, we think of introducing embedding, creating a LSTM with embedding.
The model took about 40 mins to train 4 epochs due to the size of our dataset.

Embedded LSTM (Baseline No.2, cat walking on piano)

To upgrade our model, we introduced embedding into our model. The new model starts with an embedding layer that input each data to a vector of length 60 then concatenate the original input to retain the original information. Then a dropout layer to reduce over-fitting before passing the data to LSTM layer with 512 units. Then there are two dense layer, one trained to output the note pitch and one trained to output duration of the note. The model is big and took about 10 hours to train 100 epochs, so we saved the model in .h5 format that is easier to transport and ready to be deployed. We evaluated the model with train_notes, train_durations, target_notes, and target_durations. The final loss we obtain looks very big at first glance, but we just want to focus on the note_loss, which is 3.133, much less than the loss we get from the above basic model.
This time, the music we produced has made a great improvement! We can see that the model do produce a melody, although not perfectly the harmonized music we desired.

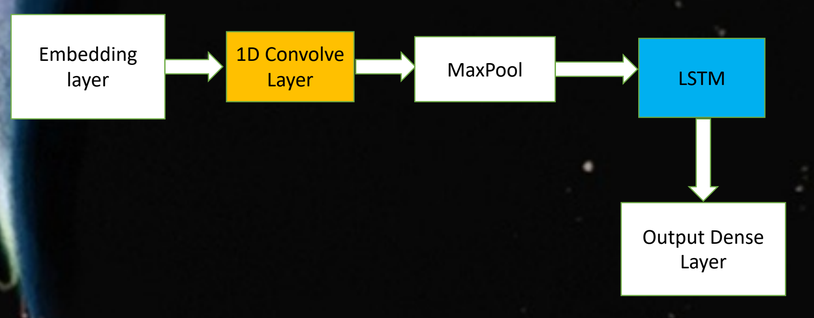


Embedded LSTM on Simplistic Dataset (On tune but still random)

We found out that the music generated often deviates from the tonal center, it goes off the key. Therefore, we think generating model without the disruption of any unnecessary information, with just the note, its relationship to tonal center, and duration, would be easier to handle and output better result. With this change, the embedded LSTM model is able to generate music in tune, but still quite random, converging on a trivial solution.

CNN LSTM (Segments of musical phrases)

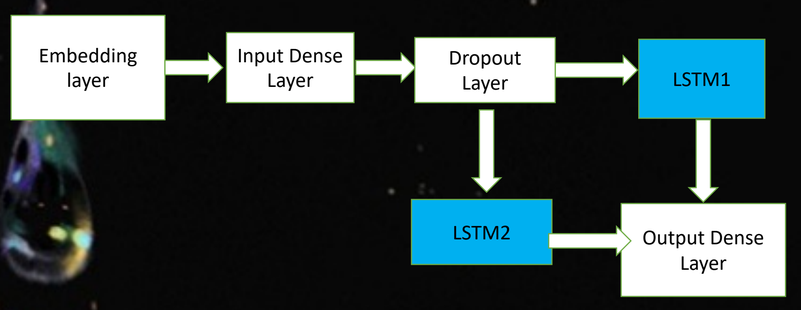
We hypothesized that convolving on music will enlarge the view of LSTM, so we applied 1-D convolution. The model contains first an embedding layer, a 1-D convolution layer, a max pooling layer, then a LSTM layer. The model took about 40 mins to train for 50 epochs on the complete labeled notes dataset and about 10 mins on the simplistic model. We observe improvements compared to the vanilla LSTM with embedding. But we see overfitting during training if we use the complete labeled notes dataset. Using a simplistic model can resolve the problem to some degree.



BiLSTM (Coherent melodies with aesthetic value)

We want the LSTM to learn faster in parallel and share the parameters among each other, so we made a model with one embedding layer, dense and dropout layer, and two LSTM layers with their outputs added, trained on simplistic data to avoid overfitting. The model took about an hour to train 50 epochs on the complete dataset and 13 mins to train 50 epochs on the simplistic dataset.

In the training process, the accuracy is relatively higher than cnn_lstm and with lower overfitting. It generates the best music compared to other models both using the complete and simplistic degree dataset, with creativity and continuity. We can see untrivial repetition of patterns in the music generated, and melody going in and out of repetitions, showing a sophisticated understanding of music.
.



Conclusion

Overall, Recurrent Neural Network, or LSTM, does not provide a promising solution to the task of Music Generation. Among all our models, BiLSTM trained on midi track data is the best, it has the highest accuracy, lowest loss, and its output is the most pleasant to listen to. The midi track provides the most complete information for the model, it contains the most latent details that cannot be captured by notes labeling, therefore, once the model has the ability to learn those hidden truths and logic, it provides the best input. Labeled notes dataset, although we put much expectation on it, gives output that converges to trivia in the end most of the time, and can cause overfitting. We noticed that embedding is quite important, likely because that the relationship between notes is not linear, so its numerical representation or scale degree alone cannot capture the relationship between each other, embedding is a much better representation. CNNLSTM was not the optimum model probably because it compresses the music too much and causes the model to over-generalize the piece. The music, we hypothesize, is very different from image data and we cannot use convolution’s equivariance property. BiLSTM, has two LSTM layer that, we hypothesize, captures different information in the music, therefore combining their result gives the best prediction.
If we have more time, we will try using more LSTM layers than two in BiLSTM, the one we have now, and test different ways of combining their results other than addition. We will also try implementing transformers and attention, or even GAN, and other state of the art generative models.