Composing Creative Music Via Automatic Note Sequence Generation

Jinho Chung (jinhoch)
Stanford University
jinhoch@stanford.edu

Euirim Choi (euirim)
Stanford University
euirim@stanford.edu

Mallika Khullar (kmallika)
Stanford University
kmallika@stanford.edu

Abstract

While natural language processing (NLP) in deep learning is well established in recent years, the field of music generation still remains premature despite that two fields share significant architectural similarities. Being inspired by the latest NLP architectures such as LSTM, Transformer, and GRU, we build novel music generation models that compose interesting musical sequences. Our systems that utilize the Maestro Dataset can be used for various human-aided compositions. We found through our experiments that our LSTM-model produced the most natural and melodic music sequences, surprisingly beating out both the GRU and transformer architectures.

1 Introduction

Music takes an important part of human culture from its inception. Different cultures at different periods of time have various ways of generating music. For many modern music compositions, we know the composers. For other musical works, especially from more than a hundred years ago, we do not know who created them. We know, however, that all music was composed by humans. It is only recently that we have attempted to build artificial intelligence systems that can generate music for us. Following this new trend, we aim to build a novel note sequence generation model that, given a note sequence primer, can creatively compose MIDI music sequences that, when played by synthesizers, sound natural and enjoyable to the human ear.

We are interested in this because all of our project group members have extensive experience playing musical instruments and sharer a love for music. We believe that this project gives us the opportunity to pair our musical appreciation with our interest in deep learning.

2 Related Work

It is currently very difficult to generate musical audio directly using machine learning and neural networks specifically due to challenges modeling long-term relationships that are common in musical compositions directly from audio. The similar challenge was apparent in natural language processing.

However, the introduction of the Transformer architecture [7] has revolutionized the field of natural language processing. The transformer's ability to remember long-term depedencies with the exclusive help of self-attention, to a degree superior to prior architectures like LSTMs and in turn RNNs, have proved to be quite effective at modeling the innate structure of human language and ensuring longer-term coherence in model output.

It is evident that music shares many of the same properties as natural language, including long-term coherence and repetition. As a result, several researchers have looked into using transformer architectures to generate music given an input of notes, usually in the popular MIDI format. Music has one key difference to natural language, however, with the relative difference of note properties

CS230: Deep Learning, Spring 2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

like pitch and timing being more important than absolute values in many cases. While capturing and using relative timing information via relative attention has been explored by some researchers [6], the original implementations were very memory-inefficient, being quadradic in sequence length. This made such implementations impractical for music composition datasets where long sequences were the norm.

By simplifying matrix computations involved in relative attention calculation, the creators of MusicTransformer [2] were able to reduce the memory requirement to be linear in terms of sequence length. This enabled them to observe dependencies in longer windows, resulting in greater coherence in music sequence output, as shown by their state-of-the-art performance on the Piano-e-Competition dataset.

Researchers from OpenAI later adopted the MusicTransfomer approach and built a system, known as MuseNet [5], that could generate musical output in particular styles and/or composers specified by the user.

3 Dataset

We trained and tested our models on the MAESTRO dataset [1] (MIDI and Audio Edited for Synchronous TRacks and Organization). The dataset contains over 1,200 MIDI files (200 hours of recordings, paired in audio and MIDI formats) accumulated from ten years of International Piano-e-Competition—the repertoire consisting of classical piano pieces.

We used M.I.T.'s Music21 library to load and parse the MIDI files into a Music21 stream object—through which we are able to extract features to feed our model. We observed the following points about the dataset: **(i)** there are a total of 1,282 MIDI files, **(ii)** Music21 was able to parse information about these files such as instruments, tracks, notes (octaves, pitch offset of each) and chords, **(iii)** A total of 7.13 million notes are played in the entire dataset and most notes tend to have a very low frequency of occurrence. The 5 most frequent <music21.note.Note> objects found played in the collection are A3, A4, D4, D5 and B4. In our data preprocessing, we chose to drop the extremely less frequently occuring notes (frequency < 10), **(iv)** Music21 also allowed us to identify the most common note offset, with the most common interval between notes in the midi files being 0.5, **(iv)** The dataset is clean and all the MIDI files contain purely monophonic melodies (single non-overlapping melody played with a single instrument), which implies the need for minimal data cleaning as opposed to our original 130,000 large dataset choice. **(v)** The compositions are primarily classical piano pieces, with the three most frequent composers being Frédéric Chopin with 201 compositions, Franz Schubert with 186, and Ludwig van Beethoven with 148.

4 Approach

4.1 Baseline

For our baseline, we used a Keras based LSTM model following the research of [4], with an LSTM neural net layer (512 nodes) dropout of 0.3, and activation layers (using ReLU as activation function and then softmax). This was trained over 50 epochs with the Adam optimizer.

To improve from our baseline, we decided to pursue three different approaches simultaneously.

4.2 LSTM-based Model

Building on our LSTM baseline, we enriched our data by unpacking notes within each chord (through Music21's chord.normalOrder which returns the normal order of the Chord represented as a list of integers) and adding those to our notes sequence. We also sped up our processes by introducing a pickle dump of the dataset of notes and chords collected after parsing our MIDI files, which were previously being computed through the Music21 library repeatedly. We also introduced a Keras ModelCheckpoint as a callback. This helped in generating .hdf5 files for weights from the network nodes from every epoch (named with epoch number and loss). In the music generation step, we were directly able to load the weights from our last run epoch, which meant that we could stop the training at any point of time, if satisfied with the loss value, and use the latest generated weights.

We also fine-tuned the LSTM to improve the baseline results by trying techniques such as representational optimisation. Instead of increasing the number of units, we made the network more complex; we added two-staged LSTM layers, where the hidden sequence of the first LSTM was provided as input to the second LSTM. We were aware that stacking LSTM layers might result in overfitting. To combat overfitting, we decided to introduce additional Dropout of 0.3 with the softmax activation.

The baseline and initial iterations of LSTM models trained on smaller subsets of the data (only some of the MIDI files from the 2011 collection of the MAESTRO dataset), faced the issue of degradation of prediction after a few bars of music, predicting similar, often the same notes repeatedly. The generated samples improved heavily as we increased the number of files we trained on, as well as the epochs we trained over. We trained the network for 200 epochs, with each batch that is propagated through the network containing 64 samples.

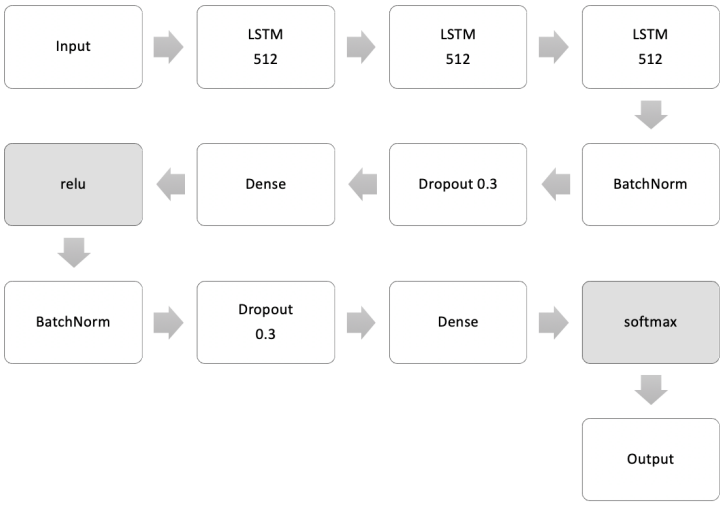

Figure 1: Network architecture for the LSTM Model

For prediction, we chose a randomised sequence from the input as a starting point for the prediction. Finally, since we added chords processing, from the resultant prediction, we added a step to identify chords (in which case we split up the Chord string into a list and created Music21's Chord object by feeding in the list of notes).

4.3 GRU-based Model

In an empirical study on GRU-based RNNs trained on polyphone sequential MIDI music, GRUs were found to outperform LSTMs in efficiency and accuracy [3], especially when long-distance relationships are not important. For our GRU model, we used the same underlying dataprocessing pipeline as we had set up for the LSTM. We trained the GRU-based network for 80 epochs, with each batch that is propagated through the network containing 64 samples. One of the initial challenges we faced in our attempt to generate music with GRU-based model was overfitting of the network to our initially relatively small training dataset. We were able to solve this issue by adding a dropout rate of 0.3.

4.4 Transformer-based Model

Our transformer model is structured as follows. We encoded MIDI using a scheme that involved binning MIDI events into 388 different so-called "note events." This binning was done using an external library. The first 128 events correpond to note-on events, one for each of the 128 MIDI pitches. There are also 128 note-off events as well as 100 time-shift events, which move forward time to the next note event. Finally, the last 32 note events are velocity events that correspond to binned MIDI velocities.

The encoding results in a sequence of numbers ranging from 0 to 387, mirroring an approach used by the authors of MusicTransformer discussed previously. Each of these numbers is then one-hot encoded into vectors. The result, after some matrix transformations, is passed into a transformer model composed of an initial positional encoder, then a transformer encoder with 4 multi-attention heads and 9 layers, each of which has a fully connected dimension size of 2048. Layer normalization is also applied to the output of each layer. The output of the transformer encoder is then passed through a linear layer and then passes through a softmax while computing the cross entropy loss.

The transformer model is structured to be a sequence to sequence model that aims to predict the next element in the input sequence. It uses an Adam optimizer for training, with weights being initialized between a narrow range (-0.1 to 0.1). Dropout with probability 0.3 is applied to every layer.

Sequences are currently generated using greedy search decoding with temperature (=1.05).

5 Results

5.1 LSTM-based Model

We trained our LSTM model for 200 epochs, with a batch size of 64 and obtained a perplexity of 5.419, suggesting our model fit the data very well. For reference, many state-of-the-art language models have perplexity scores in the single digits. Qualitatively, the 100 note samples seem promising, while a few of the 500 note samples did see some repetition towards the ends of the samples. A randomised starting point yields very different results during prediction.

You can listen to a sample generated by our LSTM model here - http://bit.ly/cs230-sample.

5.2 Transformer-based Model

After training our model for 50 epochs with a learning rate of 1e-4, a batch size of 128, and a sequence size of 96, we obtain a training set perplexity of 72.60, a dev set perplexity of 108.79, and a test set perplexity of 66.579. The training and dev set divergence, which appears more starkly in our loss/perplexity graphs, suggests some degree of overfitting.

Qualitatively, the output of the transformer is not good enough for human listening, even after 50 epochs. Notes are often repeated, which may be the result of poor model fitting as well as the limitations of greedy search decoding. With the current architecture, we do not believe that training for even more epochs will result in a better output.

5.3 GRU-based Model

We trained our GRU-based model for 80 epochs, with a batch size of 64 and obtained a perplexity of 45.604. Qualitatively, according to peers who heard the samples, the 100 note samples do have a machine-generated listening experience, and the LSTM model's outputs sound more natural and melodic.

You can listen to a sample generated by our GRU-based model here - https://bit.ly/2zeQJzJ.

6 Conclusion

In this project, we tried three approaches for composing creative music through note sequence generation.

We found that music generation using a LSTM-based model needed many epochs to be able to accurately predict next generative notes. When predicting with weights created during small epoch cycles, the model tended to generate repetitive notes towards the end of the predicted sequence. Training an LSTM-model with a deep, multi-layer architecture yielded high performance, but it took a long time to train. We tried to offset this time by also training a faster and simpler GRU-based model. Tuning the networks was harder than expected, however our initial results have been promising.

We found that a relatively basic transformer-based architecture is not easily adaptable for the task of music sequence generation, at least one that is robust to hyperparameter tuning. Possibly partly

due to maximum likelihood decoding approach we tried, transformer-based models often produced repetitive output containing many dissonant chords.

In the future, we would like to explore splitting the data into two channels (channel 1 giving the melody, channel 2 providing the chords). This would be a step towards our proposed improvement to the models: identifying more than one instrument in a stream and predicting sequences for each of them. Increasing the size of our transformer model may also be worth trying to produce high-quality music sequences.

7 Contributions

All authors contributed equally to this project. Specifically, Euirim Choi set up the underlying infrastructure and project pipelines, and worked on the transformer model. Jinho Chung worked on the literature review and research, AWS, the evaluation and a majority of the writing of the milestones and project reports. Mallika Khullar set up the baseline model and worked on the LSTM and GRU models.

8 Acknowledgements

We would like to express our special thanks to Jo Chuang, Andrew Ng and Kian Katanforoosh for meaningful instruction that helped guide our research.

References

[1] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=r1lYRjC9F7`.

[2] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer, 2018.

[3] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 2342–2350. JMLR.org, 2015.

[4] Mangal, Modak, Rahul, Joshi, and Poorva. Lstm based music generation system, Aug 2019. URL `https://arxiv.org/abs/1908.01080`.

[5] Christine McLeavey Payne. Musenet, Mar 2020. URL `https://openai.com/blog/musenet/`.

[6] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018. doi: 10.18653/v1/n18-2074. URL `http://dx.doi.org/10.18653/v1/N18-2074`.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.