IEOR 4742: Deep Learning for OR and FE

Music and Artificial Intelligence

Columbia University in the City of New York

Skand Upmanyu

Industrial Engineering and Operations Research

Business Analytics, Columbia University

su2236@columbia.edu

Abstract

Recurrent neural networks have proved to be of utmost importance in time series data predictions. In this project, we aim to go beyond by building a music generation model from a deep LSTM architecture to try to create music that has both harmony and melody and can be enjoyed by the human ear. Most of the work in this area has focused on creating short snippets of music, here we aim to create longer sequences which are continuous and have some interesting artistic effects and creativities. Our inspiration is the open-source research project, 'Magenta' [1] which focuses on music and art related projects using deep learning. In the 'Generating Piano Music with Transformer' [2] project, Magenta has trained a WaveNet model using thousands of songs for over 10,000 hours. However, we aim to train our model on a single song and try to complete the incomplete works of artists. This project deals with the generation of music using raw midi files as input using various LSTM architectures. We will specifically focus on piano music and train our models using works of famous artists like Bach, Beethoven, and Mozart. We will discuss about the challenges we faced and the ways we overcame them to generate coherent music with the help of deep learning architectures. Towards the end, we will talk about ways of improving the project going forward.

1. Introduction

Lots of research has been carried out in the field of Natural Language Processing (NLP) and tremendous success has been observed in predicting in transcription, prediction, and sentence completion. Similar has been the case with time series data where successful financial predictions have made using stock market time series analysis. Since music also entails time series data and can be comprehended in a similar fashion as words and sentences, we will use NLP analogies frequently in our project. Most of these financials and time series models are based on recurrent neural networks and LSTM architectures. We will also use similar analogy to generate new music.

This project would be different from other music generation work in this field as we will focus on creating infinitely many versions of the same song rather than generating new songs or mixing together different songs.
In this project we will try to solve the following objectives:

1. A meaningful way to represent notes in music as a sequence. We will try to convert a midi file into sequence of notes which will represent the notes that are being played at a particular time stamp

[1] Magenta: https://magenta.tensorflow.org/
[2] Generating Piano Music with Transformer: https://magenta.tensorflow.org/piano-transformer/

2. With the generated musical notes that can be comprehended by the machine, we will create a `generative model to create a song which is representative of the song on which it is trained. Moreover, we will try to create infinite versions of a single song

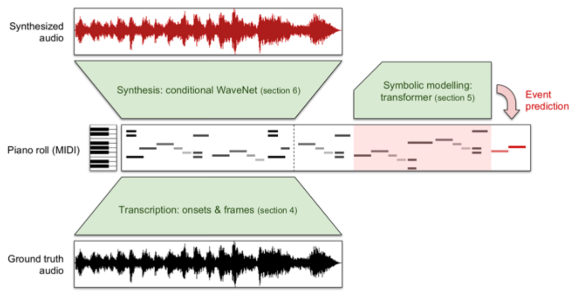
2. Background and Related Work

One of the earliest work in music generation is by Emily Howell [3] in a comprehensive study in the area of music. It is a computer program created by David Cope in the 1990s and this program was taught to compose music. By encouraging and discouraging the program, Cope attempts to "teach" it to compose music more to his liking. This is very similar to reinforcement learning. The sample and some program-composed pieces are surprisingly good enough to pass the Turing test, yet it relied on extensive manual work. David Cope began his experiments in 1981, he coded a set of rules for different style in his program that the program needs to follow in the phrase of composition. In the later version of program, he changed the program to create new output from music stored in a database, inheriting the instruction from the pieces of music in the database.

This suggests that there are two main directions to improve upon:

1. Create an encoding which can indicate that multiple notes (multiple keys of the piano) are being played at the same time
2. Create a model capable of learning long-term structure of a song so that the generated song becomes meaningful and more pleasing to humans
3. Creating different versions of the same song instead of mixing or creating entirely new songs. Such kinds of work would require generative models like GANs and generating new songs would not be possible using LSTM cells

The most significant and recent work has been carried out by Magenta mentioned previously in this report. They have used the MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization) dataset composed of over 172 hours of virtuosic piano performances captured with fine alignment (~3 ms) between note labels and audio waveforms. This dataset enables them to train a suite of models capable of transcribing, composing, and synthesizing audio waveforms with coherent musical structure on timescales spanning six orders of magnitude (~0.1 ms to ~100 s), a process they call Wave2Midi2Wave [4] which is described in the figure below.


Synthesized audio
Synthesis: conditional WaveNet (section 6)
Symbolic modelling: transformer (section 5)
Event prediction
Piano roll (MIDI)
Transcription: onsets & frames (section 4)
Ground truth audio

Transcribing piano music allows them to work with it in a symbolic form, which is both easier for training generative models and for human manipulation. From here, we got the direction of using midi files to take advantage of digital music encodings.

3. Data

Interestingly, there is huge amount of data available on the internet for midi files of piano music. We focused on famous works of Bach, Beethoven, Mozart, Burgmueller, Schubert, and Clementi. Since we are training on one song at a time, we do not require a huge repository of songs. Midi files are convenient to parse into digital sequences, we used midi files to parse them into time series data. We followed the approach of sliding windows to create multiple snippets of the same songs using which we can train our model for one timestep. The approach to create several training examples is described in the figure shown below.

[3] D. Cope, "Emily Howell," [Online]. Available: http://artsites.ucsc.edu/faculty/cope/Emilyhowell.htm
[4] Wave2Midi2Wave: https://magenta.tensorflow.org/maestro-wave2midi2wave
Image reference: https://magenta.tensorflow.org/assets/maestro/MAESTRO_models_diagram.png
Important data source: https://www.reddit.com/r/WeAreTheMusicMakers/comments/3ajwe4/the_largest_midi_collection_on_the_internet/

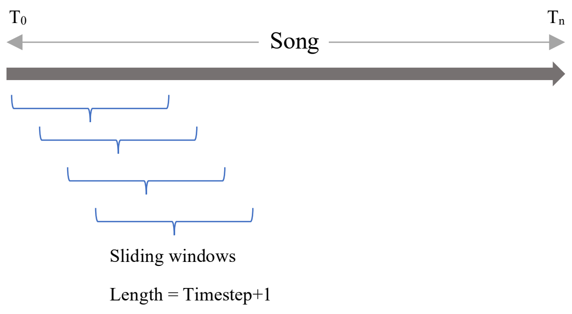


Fig. 1: Sliding window generation concept for training samples

Some examples of songs we trained our models on:

- Johann Sebastian Bach - BWV 850
- Ludwig van Beethoven - Moonlight Sonata
- Ludwig van Beethoven - Piano Sonata No. 8 (Pathetique)
- Ludwig van Beethoven - The Tempest Piano
- Johann Friedrich Franz Burgmüller – Gewitter
- Johann Friedrich Franz Burgmüller – Quelle
- Johann Friedrich Franz Burgmüller – Spinnerlied
- Muzio Clementi - Sonatina Op.36 No.1
- Christian Sinding - Frühlingsrauschen (Rustle of Spring)

All these songs were downloaded in midi format from http://www.piano-midi.de/

4. Methodology

4.a Parsing midi file

We converted midi files into a state matrix. State matrix identifies the states of the piano keys at any instant. A combination of piano keys identifies a note. Therefore, at every time stamp, we have a state matrix identifies what keys of the piano were pressed at that instant. We used the functions *midiToNoteStateMatrix* [5] to convert the midi files to state matrix and noteStateMatrixToMidi [6] to convert the output state matrix back to midi. This function converts a midi into the following shape.

Length of song x 174

where 174 is the shape of the multi-hot encoding identifying which piano keys were pressed at any instance.

4.b. Training logic

We select a size of the timestep (n) and the notes played in that timestep are the inputs. Based on these inputs, the model learns to predict the target. Our target is the musical note which is played as the next note in these timesteps. For example, if the timestep is 100 notes (n = 100), then the note played at the 101th position is our target variable. The same is depicted below:

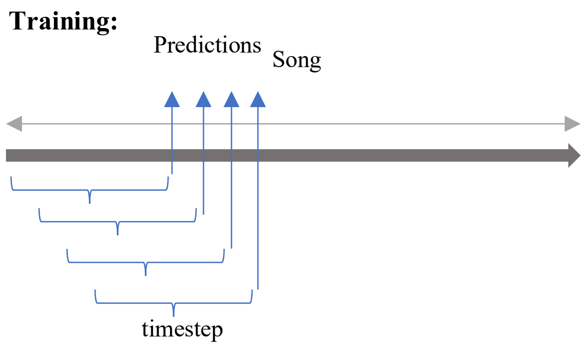


Fig. 2: Training logic

4.c. Model Architecture

We first created a vanilla LSTM architecture with the following parameters to train an LSTM cell with multiple snippets from the same song.

Attribute	Value
Input size	length of song
Output size	length of song
Hidden size	128
#LSTM cells	1

Table 1: LSTM attributes

[5, 6] Credits: Dan Shiebler - https://github.com/dshieble/Music_RNN_RBM/blob/master/midi_manipulation.py

This LSTM cell is followed by a dense layer of size 174 with sigmoid activation function to get the probabilities of each state being on.

The flow of our architecture is as follows.

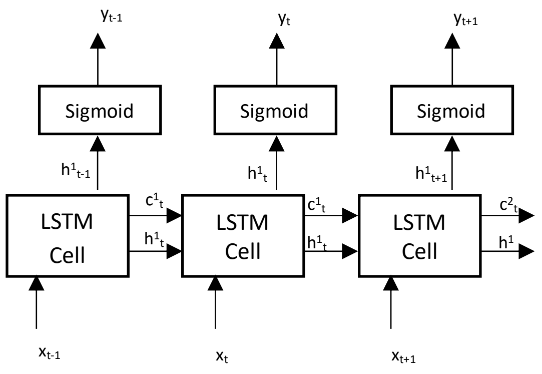


Fig. 3: Vanilla LSTM architecture

4.d. Defining loss

We use sigmoid cross entropy loss with logits to train the model. The mean of the individual note predictions is taken to get the loss across all training examples. The sigmoid cross entropy loss measures the probability error in discrete classification tasks in which each class is independent and not mutually exclusive. For instance, one could perform multilabel classification where a picture can contain both an elephant and a dog at the same time. This is true for our piano keys as well as they are not mutually exclusive. The loss is defined below

$$\text{Loss } = \max(x, 0) - x * z + \log(1 + e^{-|x|})$$

where x = logits and z = labels

For its implementation in the code, we used tf.nn.sigmoid_cross_entropy_with_logits [7] from tensorflow.

Loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits, labels))

4.e. Defining accuracy

We define accuracy in three steps:

1. Set note state = 1 for notes with probabilities greater than a certain threshold
2. Compare the generated note state to the actual note state
3. Take the average number of perfectly correctly note states across all training snippets

In our case, we tuned the threshold at 0.3 by doing several experiments and observing the results.

4.e. Defining optimizer

We used Adam optimizer to train our LSTM model using tf.train.AdamOptimizer

4.f. Hyperparameter tuning

We tuned the hyperparameters by looking at convergence time and final accuracies and loss to arrive at the following hyperparameter values.

Hyperparameter	Value
Learning rate	0.01
Training steps	2,000
Timestep	100
Stopping criteria	Acc >= 99.5%

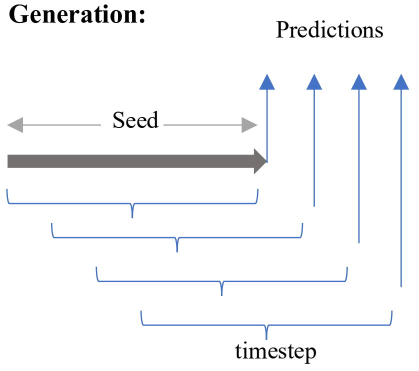
Table 2: Hyperparameters

4.g. Generation logic

During music generation, a random seed of the size of the timestep (n) is sampled from the song and the next music note is predicted from our trained model. As the next step, the generated music note and the last n-1 notes are used for generating the next musical note and process continues in a loop for the desired length of the song.

[7] https://www.tensorflow.org/api_docs/python/tf/nn/sigmoid_cross_entropy_with_logits

Generation:



5. Results and observations I

The following observations were made with the above mentioned architecture:

1. **Issue:** The model does not learn the overall structure of the song. It is only able to learn very small snippets of the song
2. **Issue:** The generated song gets stuck in a loop and produces the same sequence again and again

6. Initial conclusions

The above mentioned issues arrived because of the following:

1. LSTM cell is shallow and cannot learn overall structure of the song.
2. There is a structural flaw that needs to be fixed

7. Solution proposed I

In order to make the model deeper so that it can learn the complex features and the overall structure of the song, we tried a double-stacked LSTM by vertically stacking LSTM cells and training the model again.

The new observations and results were then observed after training.

Proposed solution:

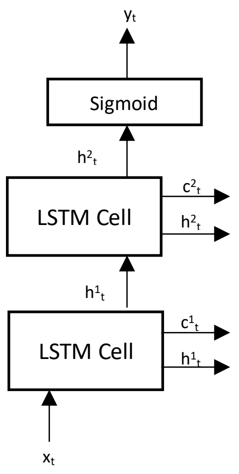


Fig. 3: Proposed solution

The new architecture looks as follows.

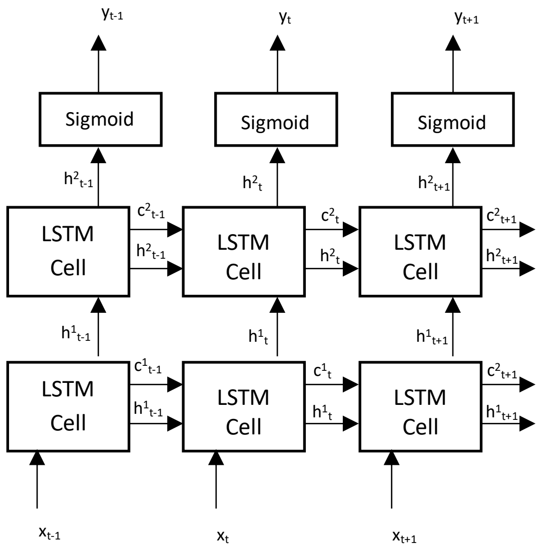


Fig. 4: Double stacked LSTM architecture

8. Results and observations II

The following observations were made with the above mentioned architecture:

1. **Issue resolved:** The model is now able to learn longer sequences without repetition. This is due to the deeper network that we built.
2. **Issue:** Most of the songs fit in this architecture but some songs with very high variance still face repetition and underfitting. This means that there is still scope to make out network even deeper for such songs.

9. Solution proposed II

The solution for underfitting of some songs was an extension of the previous solution which is adding another LSTM cell.

Since our earlier architecture was still shallow, this new architecture was expected to reduce the issue of underfitting which as observed in some songs.

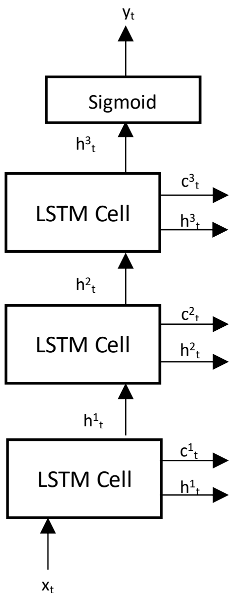


Fig. 5: Proposed solution

The new architecture looks as follows.

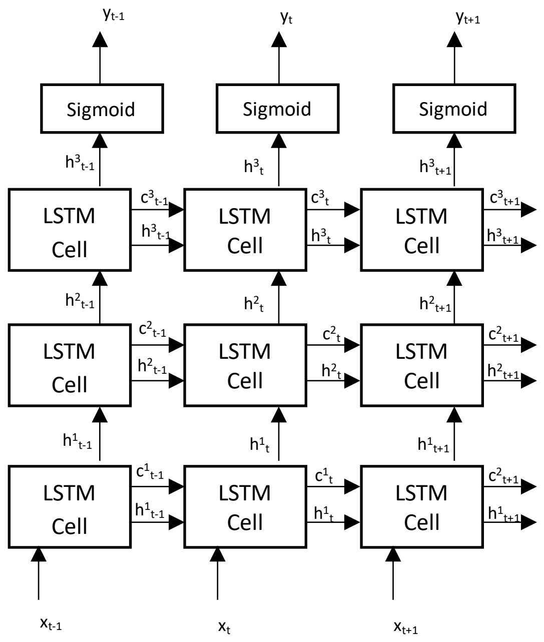


Fig. 6: Triple stacked LSTM architecture

The proposed model was expected to solve all the above-mentioned issues. However, we observed song duplication problem when implementing this solution which is discussed in detail below.

10. Results and observations III

The following observations were made with the above-mentioned architecture:

1. **Issue resolved:** The model is now able to learn longer sequences without repetition
2. **Issue resolved:** The model is now able to fit to all the songs. There is no underfitting observed
3. **Issue:** We observe duplication of song while generation. The generated song is almost a copy of the original song

The following solutions were proposed to solve the above-mentioned issues.

11. Solutions proposed III

We compared batch, mini batch, and stochastic gradient decent. Among these, the batch gradient decent worked best as it converged pretty early. But this did not fix our duplication problem.

Therefore, we had to create some new way of generating songs with the same training architecture so that infinitely many versions of the same song can be generated. For this purpose, we designed the following methods of generating songs.

11.a. Holding Bias

The idea to solve this problem was to hold on to the a certain percentage of the pattern in the seed and to let the rest change with the generation of new notes as depicted below:

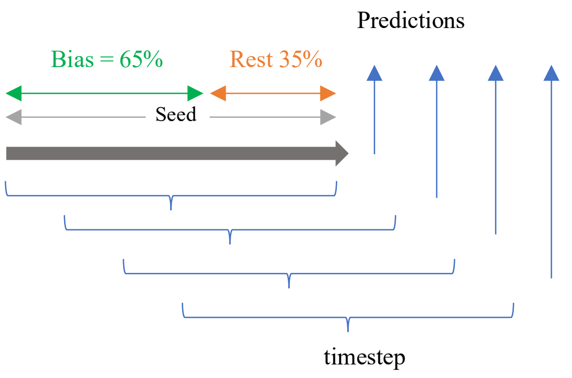


Fig. 7: Holding Bias

The idea behind holding bias is that the seed which is given initially will have an effect on the entire song which is generated as generation part holds on to that seed for the entire sequence. Therefore, every song that is generated is different from all the others.

The holding bias produced better results than previous but there was still scope of improvement. The output song initially is good but dies down after a certain amount of time.

11.b. LSTM Boosting

In this method, we try to counter the issue of the LSTM producing poor results toward the end of the song. In this method, we check the sum of the keys played in a span which is equal to the timestep. If the number of keys pressed is much lesser than the mean of the keys pressed in the seed, then we provide a fractional new seed to the model so that it can have a boost while generating the song. This makes sure that even while generating larger durations of the song, our generator produced good results. In order to maintain continuity in the song, we only input a fractional seed in order to make sure that there are no jumps in the song.

12. Results and observations IV

The following observations were made with the above-mentioned holding bias and LSTM boosting:

1. **Issue resolved:** The model is now able to learn longer sequences without repetition
2. **Issue resolved:** The model is now able to fit to all the songs. There is no underfitting observed
3. **Issue resolved:** Every generated song is now unique and coherent. The songs generated are also pleasant to hear for the human ear.

13. Conclusions

Below is our conclusion from the above mentioned experiments and research.

1. LSTM's inherent architecture is good for short sequences. In order to learn longer sequences, we need to have deeper LSTM architecture by adding more cells
2. LSTM's ability to generate long sequences is also limited. In order to generate longer sequences, we need to adopt techniques like holding bias and LSTM boosting to improve LSTM's generation capability

14. Discussion

14.a. Current challenges

Our current approach helps us generate new songs but we have not been able to find a way to quantitatively validate these results. Our conclusions and observations are based on qualitative analysis of the songs and a way to quantify these outputs would help validate these results better.

14.b. Way forward

In order to improve this architecture further, we can try creating music embeddings to include semantics of musical notes into the model. In this fashion, the model would be able to identify which two notes are similar and can be used interchangeable and at the same time, which two notes can never appear together. Such kinds of embeddings can enrich the performance of the LSTM model but it would take some research to identify the correct method of creating musical embeddings.

There is also a possibility to try various kinds of shuffling while creating mini batches while training. The right set of training batches, generated in a systematic manner can help the model converge better and faster and can ensure that we are minimizing to a global maximum.

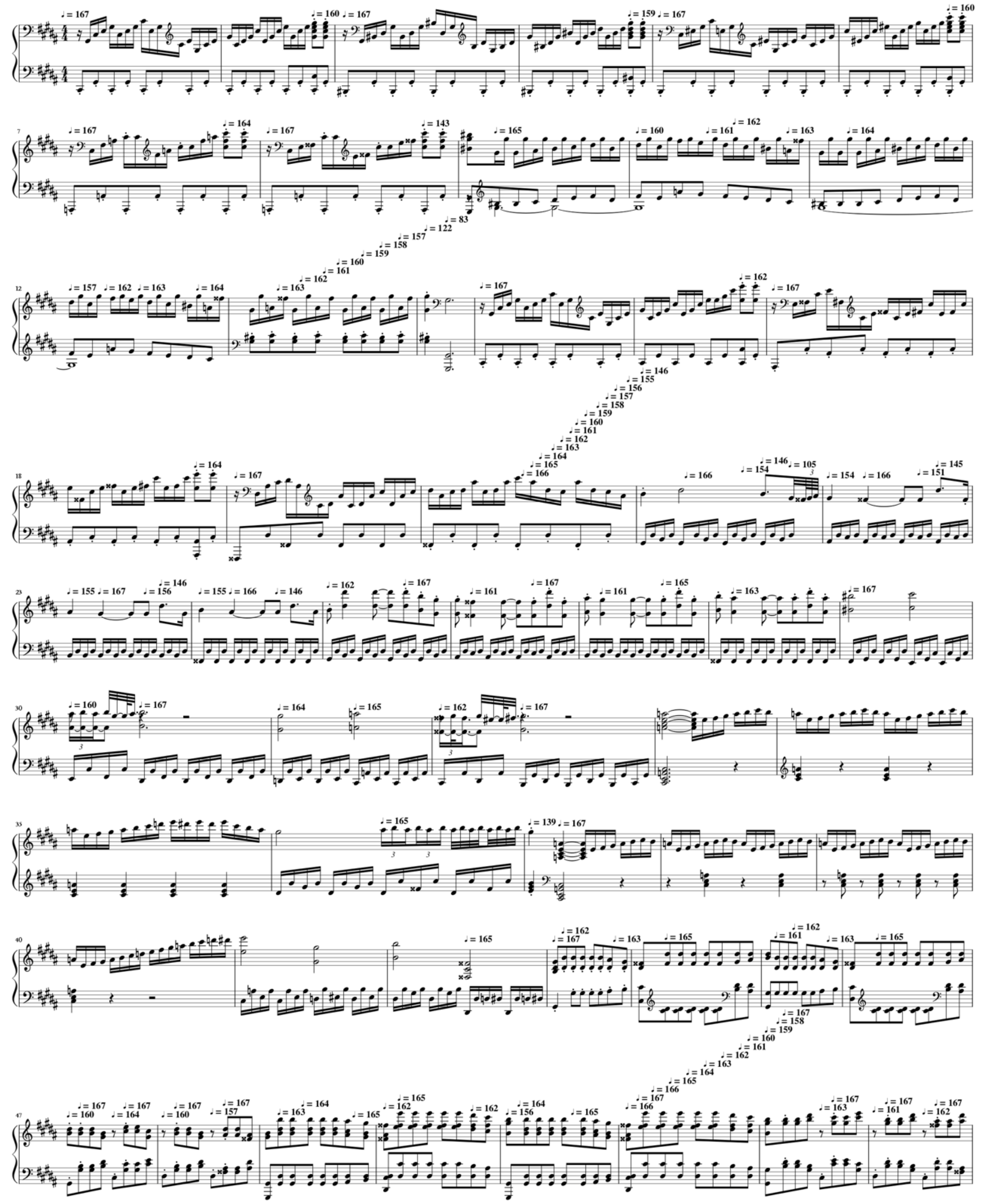
15. References

- G. H. F.-D. P. Jean-Pierre Briot, Deep Learning Techniques for Music Generation -- A Survey, 2019
- "Lakh MIDI Dataset," [Online]. Available: https://colinraffel.com/projects/lmd/
- "Musical AI MIDI Dataset," [Online]. Available: https://composing.ai/dataset
- "Million Song Dataset," [Online]. Available: http://millionsongdataset.com/
- Google, "Magenta," [Online]. Available: https://magenta.tensorflow.org/
- M. Notebook. [Online]. Available: https://colab.research.google.com/notebooks/magenta/hello_magenta/hello_magenta.ipynb#scrollTo=9YalOCM_5JP6
- D. Cope, "Emily Howell," [Online]. Available: http://artsites.ucsc.edu/faculty/cope/Emilyhowell.htm
- O. Mogren, "C-RNN-GAN: A continuous recurrent neural network with adversarial training," 2016.
- W.-Y. H. L.-C. Y. Y.-H. Y. Hao-Wen Dong, "MuseGAN," [Online]. Available: https://salu133445.github.io/musegan/

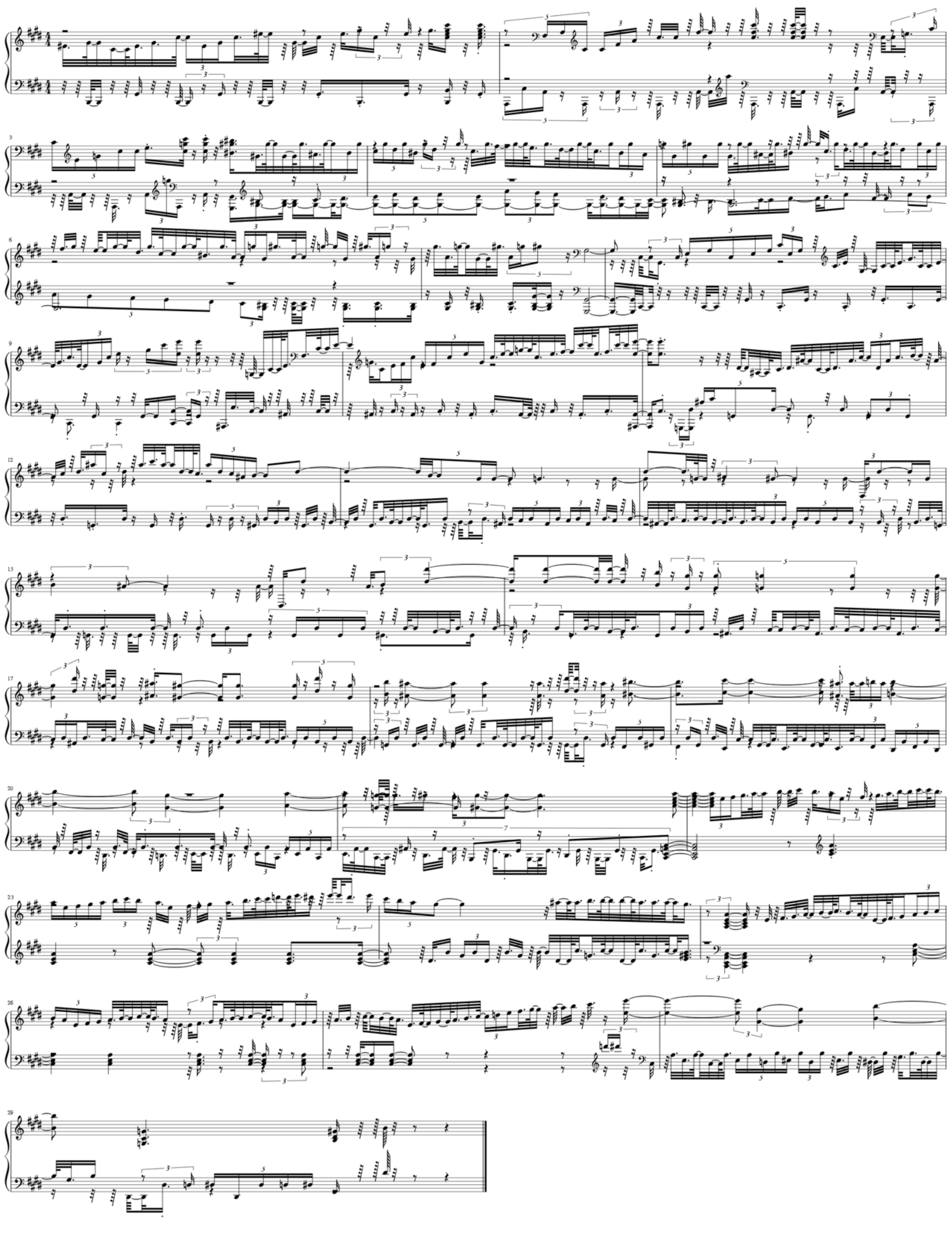
Appendix

Sample inputs and outputs

- **Original:** Ludwig van Beethoven - Moonlight Sonata



- **Generated sample 1:** Ludwig van Beethoven - Moonlight Sonata



- **Generated sample 2:** Ludwig van Beethoven - Moonlight Sonata