Generating Music with a Transformer

Paul Koesling, Maurice Cleve, Jonas Rickers

April 2024

1 Motivation

The general idea to do a project including Music Generation came from one of us discovering the Maestro Dataset [3]. When thinking about the different architectures we learnt about in the lecture, we immediately had a favorite we wanted to give a try on music.
The development of the Transformer Model has had a huge influence on Natural Language Processing. Since the first Introduction in 2017, there has been a fast development that culminated in the Release of ChatGPT in late 2022.
When we learned about the Transformer Model and how exactly it is able to pull the meaning out of a string of words using the Attention Mechanism, we thought it should be able to do well on music as well, because there are a lot of sources like [2] or [5] that suggest that in the brain, there are many similarities between the processing of language on the one hand and music on the other hand. So maybe a Transformer should also be able to work not only on natural language, but also on music.
When researching on the topic, we found that there is already so much work in the field, that we found a paper where many of the already existing systems on the topic are summarized in [6]. Also, there is already a project using Tensorflow on the topic from 2018 [4].

2 Datasets and Preprocessing

The first obvious questions is how a piece of music can be preprocessed in a way to make it usable for a transformer. In Natural Language Processing, the Input Sentence (or sentences) is split into tokens with the token having a learned representation as a vector in a high-dimensional vector space. There is some kind of natural version of this in music: The tokens can represent different time steps and the vectors can be replaced by pitches.
To not overcomplicate things, we decided to restrict ourselves to music with only one instrument and no voice. We later realized there was another reason for making this decision. We already had to find a way to make the project work with our limited computational power.

Taking piano music was a good choice then because there is a lot more potential for more things to be going on inside music besides just melody in comparison to, for example, violin. Also we assumed that there would probably be a lot of datasets for classical piano music.
At first, we thought that it should be easier to just use data from a single composer to see if the model worked. We decided to try Johann Sebastian Bach, because we knew his music was very formal. But since we did not find a suitable Dataset, we just decided to take the easy way and just use parts of the MAESTRO Dataset [3]. We tried this is in the beginning but then dropped it because it was way too large, but just taking parts of it would do the job.
We found a tutorial on the Tensorflow Website to generate music with an RNN and were confident that we could just use the same way of Data Preprocessing for our project. It cost us a lot of time to realize that we needed different tokens for a transformer and that doing it with the PrettyMIDI library like they did would not work. The next library we discovered was MidiTOK, which ended up working.
We started by importing python library we are gonna need, then we loaded the MAESTRO Dataset and selected a part of it, since using the whole dataset would take way too long in training. To make training easier, we restricted ourselves to pieces of a certain length and only used pieces with a measure of 4/4. Then we put all the MIDI Files we want to use into a list.
For tokenizing, we could just use REMI from the MidiTOK library. We also used the Byte-Pair Encoding from the MidiTOK library, which helps representing a large vocabulary with a small size of units. We then saved the tokenizer using the pickle library to use it later when we want to turn the tokens back into MIDI files to listen to. Then we applied the tokenizer to the MIDI files in the list we created earlier.
We then built a python class in which the tokens are first extracted from the MIDI files and then saved in a dataset with the usual batching and prefetching. Then we only have to seperate the data in a training and a validation dataset.

3 Model Architecture

The Transformer Model has been introduced first in [1]. The important difference to prior models is the Attention Mechanism, allowing the Transformer to gather information on tokens no matter the distance in O(1). This is very good for analyzing and creating music because there can be a lot of different connections between totally different points in a piece.
For example, when there is a key change in the middle of a piece and then in the end, the piece returns into the key it started with, there is some kind of logical connection between the start and the end, and the Transformer can capture that no matter the distances. One can find sources like [7] where a lot of different interpretations of music are suggested, so there are a lot of connections the Attention Mechanism can pick up.
We knew we wanted to use the Architecture from [1], the only thing we im-

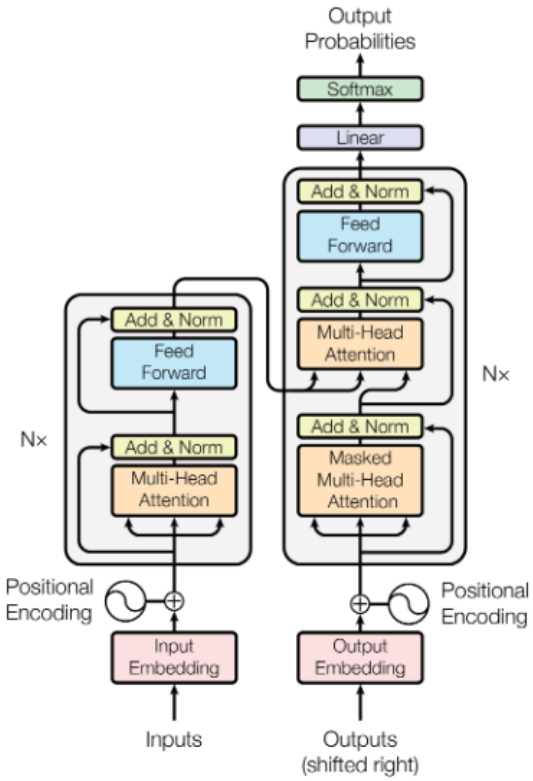

Figure 1: The Original Transformer Architecture

mediately changed was using the SiLu Activation function, which is defined as

$$SiLu(x) = x \cdot \text{Sigmoid}(x)$$

A lot of sources seemed to combine the Layer Normalization in the Original Transformer with Dropout, and since the lectures showed us how Dropout worked, we were curious if it could be useful for our model, and it was. We tried to make the code as efficient as possible my maximizing the amount of parallelization done.
For example, instead of seperating the Input Embedding and the Positional encodings, as it looks like in the Original Attention Paper, we did put both into a single class where everything needed is already done.
One thing we actually did not get to work in the end was to skip the first token the Transformer has to receive and just generate something out of nothing.

4 Training Procedure

The core principle underlying the training of a transformer model for music generation, or indeed for modeling any type of sequential data, revolves around the concept of next-token prediction. In this process, the transformer is trained with thousands of examples of token sequences. Its performance is evaluated based on its ability to predict the subsequent token in these sequences. This method does not require separate source and target datasets since any input sequence can be readily transformed into a corresponding target output.
To prepare the data for training, each sequence of tokens, of a length defined by the hyperparameter seqlen, is duplicated. The duplicate is then altered by shifting all tokens one position to the left and discarding the initial token. This shifted sequence serves as the target, while the input sequence is formed by removing the last token, thus aligning the indices for input-output pairs.
In contrast to simpler models, such as bi-grams where the prediction is based solely on the immediately preceding token, the transformer considers the entire preceding sequence up to the current token. This means that for each target index ii, the model takes into account all preceding inputs up to ii, starting from the very first token. The length of these input subsequences can vary from 1 up to seqlen, resulting in each training sequence containing as many training examples as its length.
For practical purposes, and to balance computational efficiency with the richness of information in each training example, we have chosen sequence lengths of either 2048 or 4096 tokens. Considering the computational demands, we have allocated substantial resources, purchasing 200 compute units on Google Colab, to ensure that training proceeds within reasonable time frames.
The output from the transformer is a probability distribution over all possible tokens, and the model's learning is guided by minimizing the categorical cross-entropy loss. Gradient descent is then applied primarily to the decoder block and the final dense layer to update the model parameters

5 Generating Music

Upon successful training, the transformer model is capable of generating piano music. The generation process initiates with the model receiving an input sequence of tokens, which could be empty at the start. Based on this input, the model predicts a probability distribution for the next token. Rather than selecting the most probable token deterministically (a method known as greedy selection), the model samples from this distribution.
This stochastic approach to token selection ensures that the resulting music is not only varied but also rich in its musical texture, reflecting a more dynamic and realistic composition. Such a method helps in generating music that is not only novel each time but also pleasant and engaging, capturing the complexities of human-composed music.

6 Project Discussion

Luckily, one of us studied Math before doing his Master in Cognitive Science, so understanding how the Transformer Model functions was not the hardest thing. Even though the Transformer is more complicated than other forms of Neural Networks like CNN, it is still based on many Matrix Multiplications with some Projections and Activation Functions in between.
This showed us how much more there should be to discover in the Field of Machine Learning, not only for Music Generation, because there is so much to be discovered and so much to try out from the field of Math.
One thing we underestimated was how far it would be to get good data and make it usable. We thought it should be easy to just obtain some MIDI Data, transform it into two-dimensional tensors and put them into the Transformer, but we tried out probably a two-digit-number of ways to convert the MIDI Files before finding something that actually works.
We felt like what we did with our Transformer Architecture is only the tip of the iceberg of possibilities in Automated Music Generation. What would really help this field would be an easier way to make all kinds of musical data usable in the way we did, since this is probably a thing keeping many of people from being interested in trying their luck in this field.
Also, the possibilities should get much higher with more computational power, because not only one could build a bigger model, but also take in more complicated pieces with more instrument. We are really looking forward to the development in the field regarding this, because AI trying to generated orchestral music with big numbers of instrument should be a really interesting topic in the future.
Getting back to the idea that there should be potential for more complex models in the future, there should also be more to do with the models we have right now and the computational power rising.
Sound data, especially in high-quality formats, is voluminous and intricate. High-fidelity audio files are considerably larger than most text files or even many types of image files. This size difference leads to greater demands on storage and processing power.
In the case of MIDI files, while the data is more compressed and abstract, it lacks the richness of raw audio files, capturing only the information about musical notes and their timings rather than the audio's timbre and quality.

There is a noticeable scarcity of available musical datasets, particularly those that are well-labeled. While the internet is replete with freely accessible music, the volume of available music data pales in comparison to the repositories available for images and text. Labeling music accurately requires a nuanced understanding of musical composition, often necessitating the involvement of trained musicians.
This is a stark contrast to the labeling of images, which can be accomplished with minimal training. The expertise required for music labeling not only makes the process costlier but also limits the availability of high-quality, labeled musi-

cal datasets. For instance, Google's MusicLM dataset, which is considered state-of-the-art in music generation, includes only around 5,500 labeled tracks—a minuscule amount when compared to image databases.

Producing high-resolution sound through neural networks remains a formidable challenge. This is true whether the sound is musical, speech, or any other audio type. Although technological advancements have made significant strides, achieving a level of quality that rivals natural sound production still requires extensive computational resources and sophisticated modeling techniques. Temporal Complexity of Music
Music's temporal nature further complicates data labeling and model training. A single piece of music can encompass multiple themes and emotions, which may not be fully captured by a single label. For instance, a song labeled as "joyful" might contain passages that evoke a sense of sadness or nostalgia, which aren't reflected in the label. This mismatch can mislead the training process of models, potentially leading to less accurate or inconsistent outputs. Broader Implications
The challenges extend beyond the technical aspects to include the practical application of generated music. Ensuring that the music produced by AI systems is not only technically sound but also emotionally resonant and contextually appropriate is a significant hurdle. Moreover, the legal and ethical implications of using AI in music production, such as copyright issues and the authenticity of creative outputs, are ongoing concerns.
In conclusion, while the potential for artificial music generation is vast, the field faces substantial hurdles that stem from the unique properties of audio data, the scarcity and quality of datasets, and the computational demands of processing such data. Addressing these issues effectively will require not only more sophisticated technologies and methodologies but also a collaborative effort to amass and curate high-quality, diverse musical datasets.
One thing that only came to our mind when we were close to finished was the idea of not taking midi data, but audio signals as an input. There is actually a counterpart to this in the first use of transformers in Natural Language Processing, because there we could also give an input of an audio file with someone speaking instead of giving a text as input.
Of course, there are already methods to just extract the words out of Natural Language or to extract notes out of musical pieces, but a more interesting approach in the future might be to skip that part and just try to give some kind of mathematical representation of the wave in the audio signal and learn on that. Since a continuous wave is just a continuous mathematical function $f : \mathbb{R} \to \mathbb{R}$, we can use mathematical properties of those functions.
A property that might be very useful is the fact that such a function f can be approximated arbitratily well both by polynomials with coefficients in the real numbers which are functions of the form $\sum_{i=0}^{n} a_n x^n$ and by function of the form $\sum_{i=0}^{n} cos(nx) + sin(nx), n \in \mathbb{N}$. "Arbitratily well" means that if have a given maximum error of ϵ, there exists an $N \in \mathbb{N}$ so that if we take the sum until that N, the error is a maximum of $\epsilon \forall x \in \mathbb{R}$.

For the case of audio signals, it should work even better because there is a finite amount of different values to plug in for x, which is the sampling rate (mostly 44100) multiplied with the amount of seconds the piece has.
So in principle, one can use those functions to approximate the waves, then use those as a representation and just give those to a machine learning model. The big difference in this approach would be that this kind of representation is not the way that we as humans perceive music or natural language, or at least, we think that the way that ChatGPT gets natural language and our model gets token is the way that our brain perceives them.

References

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

[2] Mireille Besson, Eva Dittinger, and Mylène Barbaroux. Music in the brain. 08 2018.

[3] Adam Roberts Ian Simon Cheng-Zhi Anna Huang Sander Dieleman Erich Elsen Jesse Engel Douglas Eck Curtis Hawthorne, Andriy Stasyuk. Enabling factorized piano music modeling and generation with the MAESTRO dataset. 2019.

[4] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, and Douglas Eck. Music transformer: Generating music with long-term structure. *arXiv preprint arXiv:1809.04281*, 2018.

[5] Lutz Jäncke. The relationship between music and language. 04 2012.

[6] Hanwei Liu Junwei Pang Yi Qin Qidi Wu Lei Wang, Ziyi Zhao. A review of intelligent music generation systems. 11 2023.

[7] Geraint A. Wiggins. Music, mind and mathematics: theory, reality and formality. 2012.