EE 641 – Deep Learning Systems

Music Generation using Transformers

Divya Nandlal Sahetya, Krishna Dheeraj Krovi, Vaishnavi Channakeshava

Tuesday 13th December, 2022

1 Abstract

To generate music using musical notes in a non-trivial way, unlike generating it from the actual sound waves. We propose to create a model to generate music based on a few initial notes from the music MIDI file provided (Figure 1). The basic principle is to build a sequence model for music. That is, take an input sequence and predict a target sequence. We put forward a solution based on transformers to modify music notes to tokens in an auto-regressive manner as a part of a single input stream. The traditional approach of music generation using Recurrent Neural Networks has to learn to proactively store elements to be referenced in a fixed-size state or memory, making training potentially more difficult. To optimize this problem we are using concepts of transformer [1] that can help capture the multiple levels at which self-referential phenomena exist in music. The project is implemented with a MLOps fashion.

Figure 1: Music Notes

2 Introduction

Music is often composed of repeating elements at different levels. It is an arrangement of sound that has been purposefully structured at various time and frequency scales to convey a range of ideas and emotions.

Repetition and long-term structure are key considerations in making a musical work coherent and understood. To generate music, the model should be capable of learning long term memory and be able to reference elements even in the distant past. In the past works we have seen models like Recurrent Neural Networks (RNNs) and Long-Short Term Memory(LSTM) used to generate music. With self-attention and positional encoding in transformers, it has been observed that they work well in generation of music. The music notes is converted to time-ordered sequence of tokens such as events. Transformers [1] can then be applied to model the probability distribution of the event sequences, and to sample from the distribution to generate new music compositions. Transformers have achieved better results in many tasks that require maintaining long-range coherence [2]. This approach has been shown to work well for composing minute-long pieces of classical piano music with expressive variations in note density and velocity [3], in the format of a MIDI file.

The original Vanilla transformer references through self-attention; however, because it is based on absolute positional encoding, it has difficult to keep track of the repeated tokens based on the relative distance between the tokens, orderings, and periodicity. Thus, transformers model with relative attention concentrates more on attention based on how far apart two tokens are. The model is able to focus more on relational features, which directly adjust attention based on how far apart two tokens are from one another. Relative self-attention also allows the model to generalize beyond the length of the training examples, which is not possible with the original Transformer model. This is practical for long sequences such as musical compositions since their time and space complexity of self attention grows quadratically in the sequence length.

In our project, we have pre-trained our Transformers model with relative position encoding using Maestro dataset and fine-tuned the same model on Tegridy MIDI Dataset [8] to obtain a better performance in music generation. It can generate a minute-long (thousands of steps) compositions with compelling structure, continuations that coherently elaborate on a given motif. Musicians could utilize this model to generate background scores, tunes, and songs which they want to extend from a few notes that come out of their mind.

There is a pipeline including steps such as data fetching, preprocessing, training, storing the best hyperparameters. When required to retrain, the parameters to be configured are made accessible and will be able to pass as arguments to the script.

3 Related Work

- Peter Shaw, et al., uses Transformers with relative positional self-attention. It states that relying entirely on an attention mechanism, the Transformer introduced by [1] achieves state-of-the-art results for machine translation. In contrast to recurrent and convolutional neural networks, it does not explicitly model relative or absolute position information in its structure. Instead, it requires adding representations of absolute positions to its inputs. In this work they have presented an alternative approach, extending the self-attention mechanism to efficiently consider representations of the relative positions, or distances between sequence elements. [3].

- Cheng-Zhi, et al., demonstrates that a Transformer with our modified relative attention mechanism can generate minute-long compositions (thousands of steps, four times the length modeled in Oore et al., 2018) with compelling structure, generate continuations that coherently elaborate on a given motif, and in a seq2seq setup generate accompaniments conditioned on melodies [2].

4 Dataset

We have used two datasets: MAESTRO[V2.0.0] [9] and Tegridy datasets [8] for this project. For pretraining, we have used MAESTRO and Tegridy for generating music i.e. fine tuning task. MAESTRO [V2.0.0] dataset is composed of 200 hours of piano performances taken from Piano-e-Competition of 1292 performances (files).

We have used the Tegridy-Piano-Transformer-Dataset-CC-BY-NC-SA in the collection of Tegridy datasets consisting of 6500 MIDI files. This dataset has been crowdsourced as part of Project Los Angeles and some source compositions have been produced with OpenAI's MuseNet.

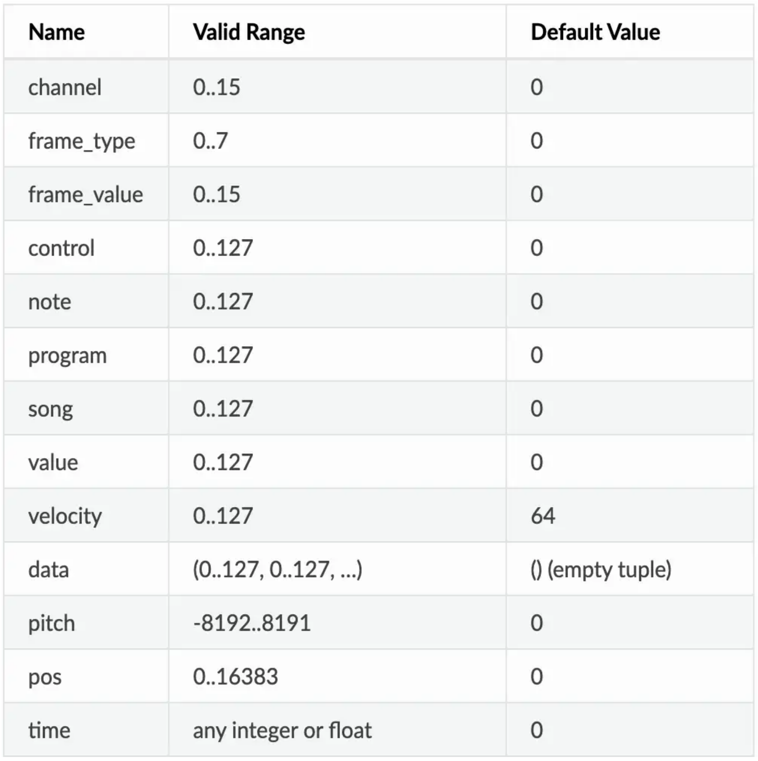
The format of the files used is MIDI (Musical Instrument Digital Interface). Unlike digital audio (.wav) files, compact discs, or cassettes, MIDI does not capture and store actual sounds. It is mostly used to specify which musical notes to be played when, for how long, at what volume, and with which ”instrument”. MIDI files are very much smaller than other audio files. The compact size of MIDI files makes them especially well suited for delivery over the Internet.

5 Data Preprocessing

The dataset is first split into train (75%), test(13%) and validation(12%) as a part of the preprocessing step. For the next part of the preprocessing we have used the pretty midi python library for extracting information from MIDI files. This module has the functionality to store all the events that constitue a music piece, the timing information, meta events, and utility functions for manipulating, writing out and inferring information about the MIDI data it contains. A MIDI file consists of events and each event consists of two components: a MIDI time, and a MIDI message. The midi time/message pairs are sent as a sequence of bytes. The main commands in a midi file are Note-on, Note-off, Time-shift and velocity. The number of bits assigned for Note-on is 128, Note-off is 128, timeshift is 100 and velocity is 32 bytes. The output of an encoded MIDI file is a list of integers where each integer is a sum of bits represented for Note-on, Note-off , timeshift and velocity. The encoded info of each MIDI file is then saved as a .pickle file to further use for training.
Details of each of the fields:

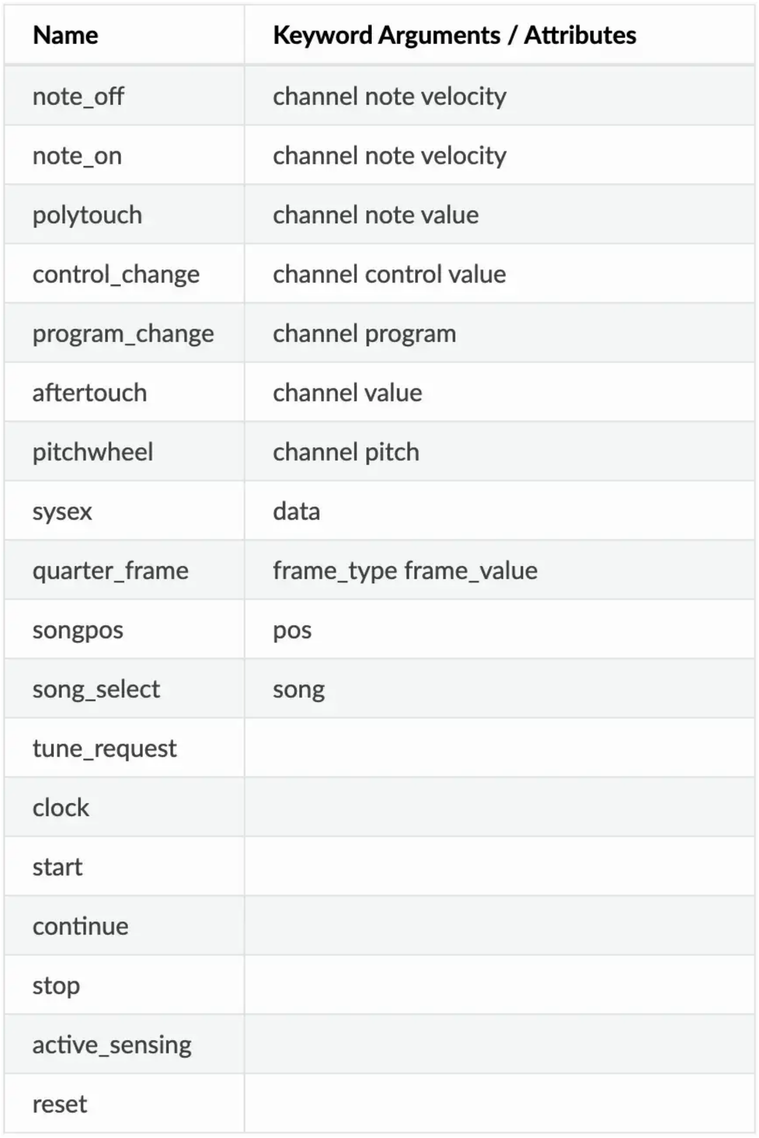
- “note_on” tells the key is to be pressed (or released, if velocity=0).
- “note_off” tells the key is to be released (velocity should always be set to 0).
- “channel” tells to which channel the sound is to be sent. The standard midi supports 16 channels simultaneously.
- “note” tells which key it is. The map (Figure 3 and 2) below mentions the details for each key on piano keyboard to each midi note id.
- “velocity” tells how fast to strike the key, the faster it is, the louder the sound is.
- “time” specifies the waiting time between the last and current operation. The duration of a note is the sum of “time” from each message in between of 2 nearest messages about the same note, where the first one specifies the note (when “note_on”, and “velocity” ¿ 0) and the last one specifies the note (when “note_off”, or “note_on” with “velocity”=0).

Below is the summary of message type and parameter value range:



Name	Valid Range	Default Value
channel	0..15	0
frame_type	0..7	0
frame_value	0..15	0
control	0..127	0
note	0..127	0
program	0..127	0
song	0..127	0
value	0..127	0
velocity	0..127	64
data	(0..127, 0..127, ...)	() (empty tuple)
pitch	-8192..8191	0
pos	0..16383	0
time	any integer or float	0

Figure 2: Message type and parameter value range



Name	Keyword Arguments / Attributes
note_off	channel note velocity
note_on	channel note velocity
polytouch	channel note value
control_change	channel control value
program_change	channel program
aftertouch	channel value
pitchwheel	channel pitch
sysex	data
quarter_frame	frame_type frame_value
songpos	pos
song_select	song
tune_request	
clock	
start	
continue	
stop	
active_sensing	
reset	

Figure 3: Message type and parameter value range

6 Implementation

The steps involved in this project are as summarized below:

1. Step 1 : Data Collection :
   - Train Dataset: Maestro V.2.0: is used in this project. The dataset contains about 200 hours of paired audio and MIDI recordings from ten years of International Piano-e-Competition. Size of the maestro-v2.0.0-midi.zip dataset is 57MB (85MB uncompressed).
   - Test Dataset: Tegridy-Piano-Transformer-Dataset-CC-BY-NC-SA: The dataset consists of 6500 midi files and the size of Tegridy-Piano-Transformer-Dataset-CC-BY-NC-SA.zip is 10.7MB (17MB uncompressed).
   - A train/validation/test split configuration is also proposed, so that the same composition, even if performed by multiple contestants, does not appear in multiple subsets.
2. Step 2 : Data Preprocessing / Tokenization : The dataset is first split into train (75%), test(13%) and validation(12%) as a part of the preprocessing step. For the next part of the preprocessing we have used the pretty midi python library for extracting information from MIDI files.
3. Step 3 : Baseline Model Training: The baseline model is trained on the Maestro dataset for 150 epochs and for time sequence length=512. The best weights of the model is saved in pickle format on validation.
4. Step 4 : Baseline Model Evaluation : Subsequently, the analysis of the training is done using validation accuracy and loss metrics.
5. Step 5 : Retrained model Training : The baseline model is trained on the Maestro dataset for 150 epochs and for time sequence length=512. The best weights of the model is saved in pickle format on validation.
6. Step 6 : Retrained model Evaluation : Subsequently, the analysis of the training is done using validation accuracy and loss metrics.

7 Model Selection and Hyperparameter Tuning

The best model is chosen on the basis of the accuracy on the validation set.

Algorithm 1 Model and hyperparameters

Require: Train, validation and test data is available with below preprocessing:

- Music in midi file format.
- The musical notes encoded using pretty-midi encoding.
- params =
  - optimizer = Adams Optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon = 10^{-9}$, warmup_steps = 4000, the learning rate is increased linearly for the first warmup_steps training steps, and decreasing it thereafter proportionally to the inverse square root of the step number.
  - *batch_size* = 4
  - max_sequence = 512
  - n_layers = 6
  - num_heads: 8
  - d_model: 512
  - dim_feedforward: 1024
  - (Regularization) dropout: 0.1
  - epochs : 150
  - loss : CrossEntropyLoss

```
function TRAIN(hyperparameters, train_data, val_data)
    best_score = 0
    best_weights = {}
    for epoch in epochs do

        model.train()
        model.eval()
        curr_score = model.
        if $score \leq best\_score$ then
            best_score = curr_score
            best_weights = model.state_dict()
        end if
    end for
end function
```

8 Results

8.1 Baseline Model

Baseline model is a Musical Transformer which is a PyTorch implementation of Transformer with a relative positional encoding. This is proven to have better performance [3] on longer sequences. The model in the project is trained on the MAESTRO Dataset with a train-val-test split of 75-13-12 for a maximum sequence of 512 for 150 epochs. The plot for training obtained is as shown in Figure 4.

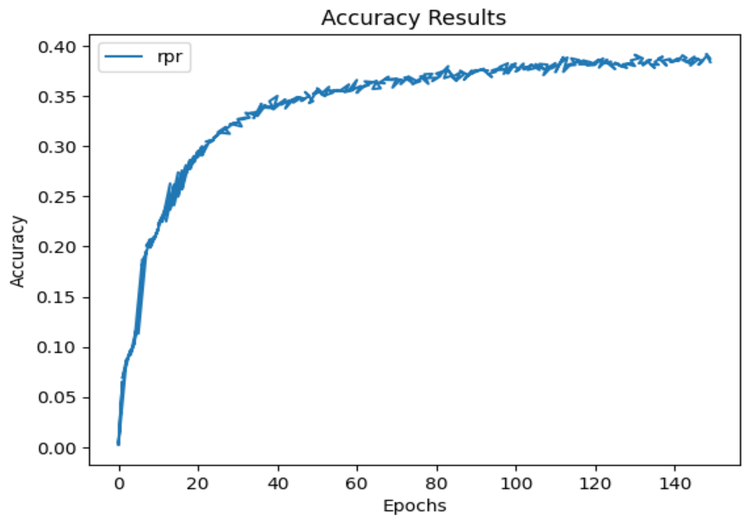


Figure 4: Baseline Model Accuracy Curve (token length=512, epochs=150)

Cross Entropy Loss was used to compute the difference in the token generated by the model and the actual value of the encoded note at that point. A curve showing the loss per epoch was plotted and obtained as shown in Figure 5. The results obtained are as below.

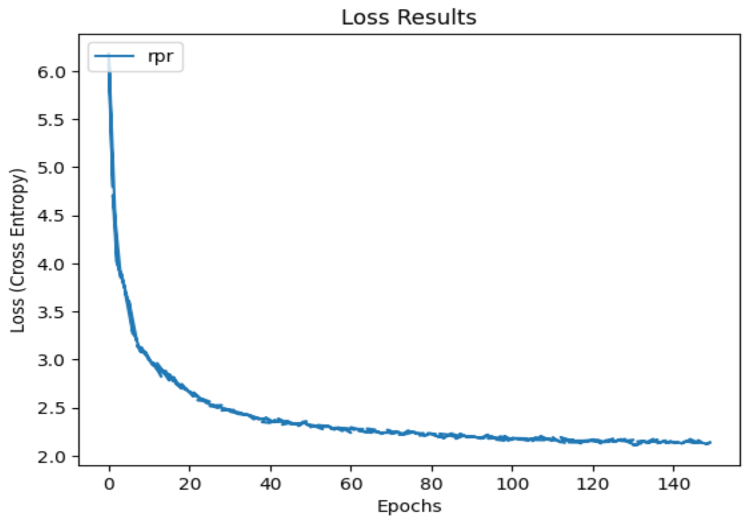


Figure 5: Baseline Model Loss Curve (token length=512, epochs=150)

Learning Rate was dynamic and adjusted with a scheduler with each epoch. It increases linearly in the beginning and then decreases as the model starts to perform better. The plot of learning rate with epochs is as shown in Figure 6.

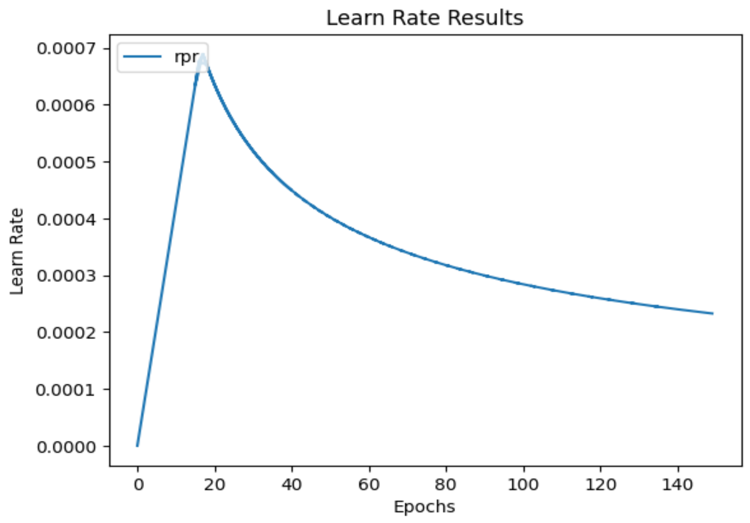


Figure 6: Baseline Model Learning Rate Curve (token length=512, epochs=150)

- Best validation accuracy: 0.3919921875 for epoch 148
- Best validation loss: 2.1082846138212417 for epoch 130

Performance on Test Data

Performance of the model on the test data was done by taking a random midi file from the test set and generating a primal (motif) from it. Then this is sent to the model to get the output sequence for the desired sequence length. The plot for the Primer midi is as shown in Figure 7.

At training time, seq2seq approach as a sequence labeling problem. Symbols labels of what the next symbol is assigned as the target sequence. From this, accuracy and loss is computed, simply as a proportion of cases when a correct label is assigned.

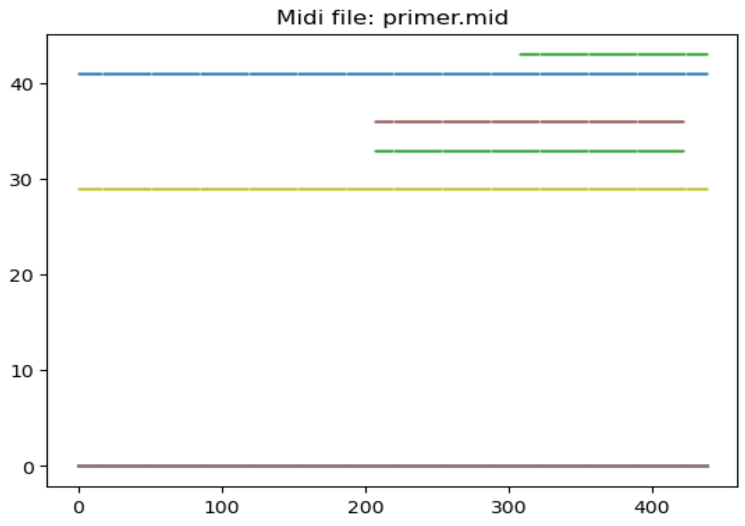


Figure 7: Baseline Model Output for Test data Primer midi

A plot of pitch against time to see how close to actual file is the model generating is as shown in Figure 8.

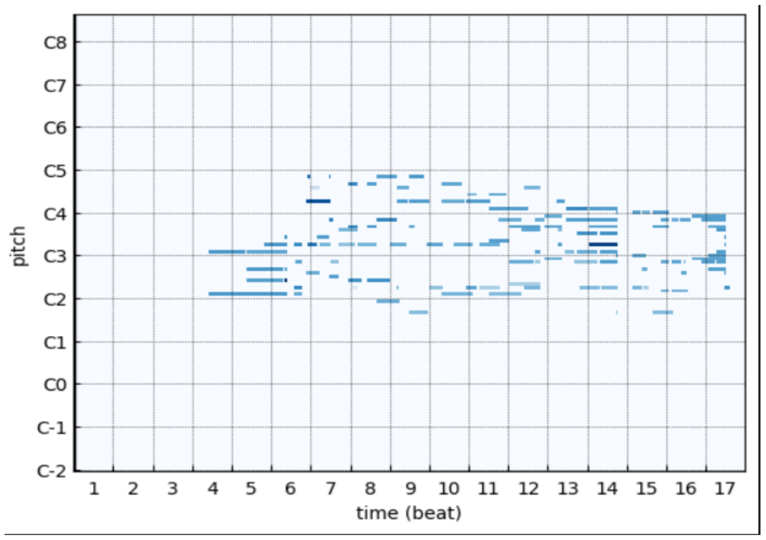


Figure 8: Baseline Model Generated Music (pitch vs time)

A more visually familiar plot, like in an equalizer is also generated as shown in Figure 9.

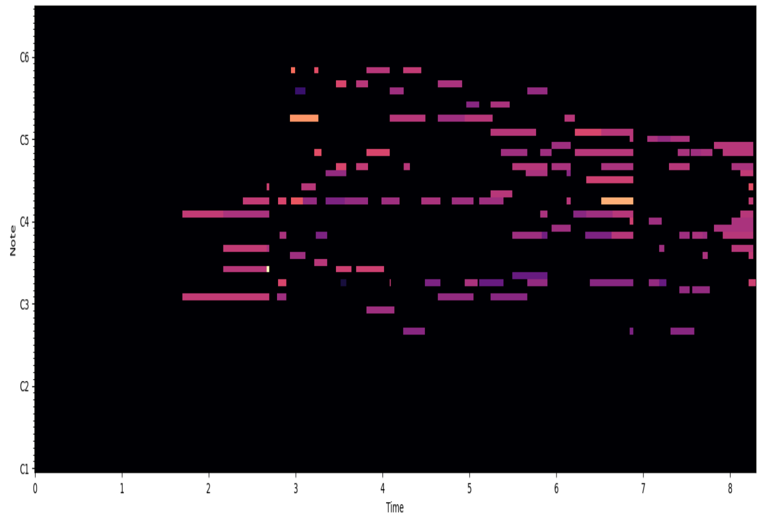


Figure 9: Baseline Model Generated Music (notes vs time)

A same plot giving us more information about the notes and their on-off times and the velocity at a glance is as shown in Figure 10.

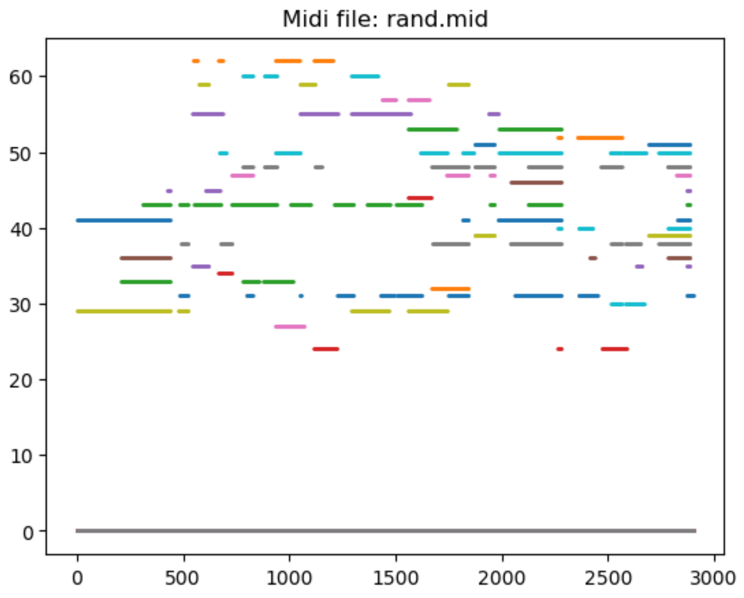


Figure 10: Baseline Model Generated Music file

8.2 Retrained Model

The baseline model was trained on the new custom dataset Tegridy for 50 epochs. The accuracy values were plotted and obtained as shown in Figure 11. First the 6500 samples were preprocessed and encoded, then the transformer was trained with initial weights of 150th epoch of the baseline model.

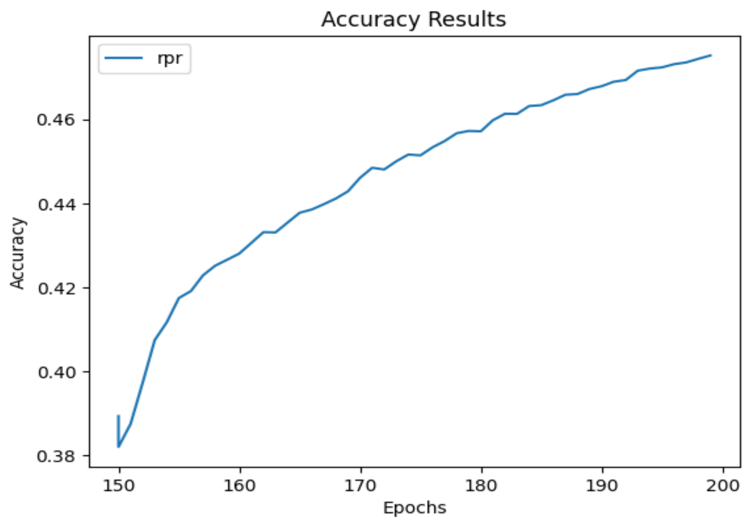


Figure 11: Retrained model Accuracy Curve (token length=512, epochs=200)

The loss curve for the retrained model decreases to 1.7 with further training and is plotted as shown in Figure 12.

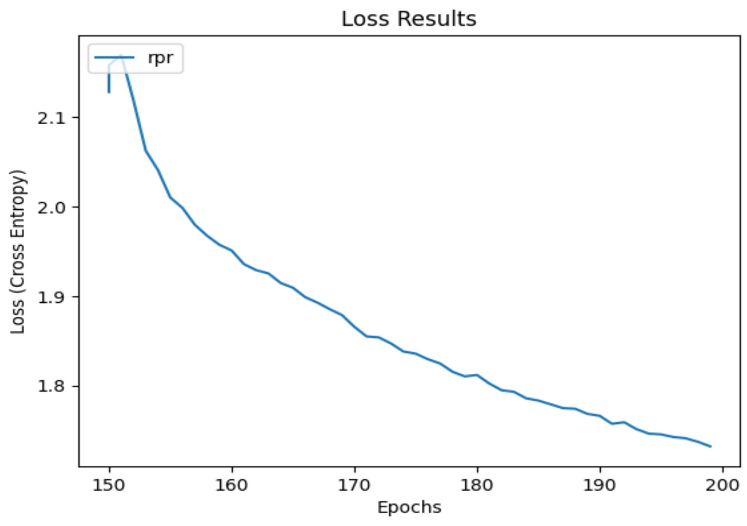


Figure 12: Retrained model Loss Curve (token length=512, epochs=200)

The learning rate vs number of epochs for the retrained model is plotted in Figure 13

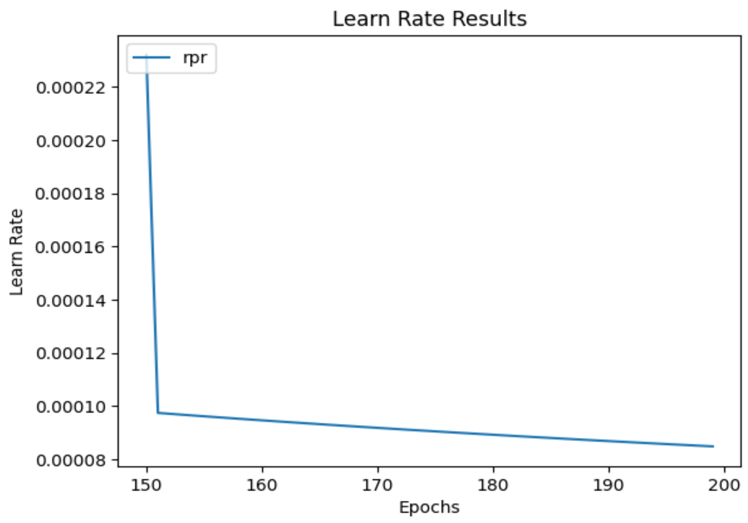


Figure 13: Retrained Model Learning Rate Curve (token length=512, epochs=200)

For the Retrained model the best validation accuracy and loss results are as shown below.

- Best validation accuracy: 0.4756218934094256 for epoch 200
- Best validation loss: 1.7286721515374612 for epoch 200

For computing the performance on the test set, we choose the primer from the test split, preprocess it and send to the model. The length of the output sequence is also set and for the output generated

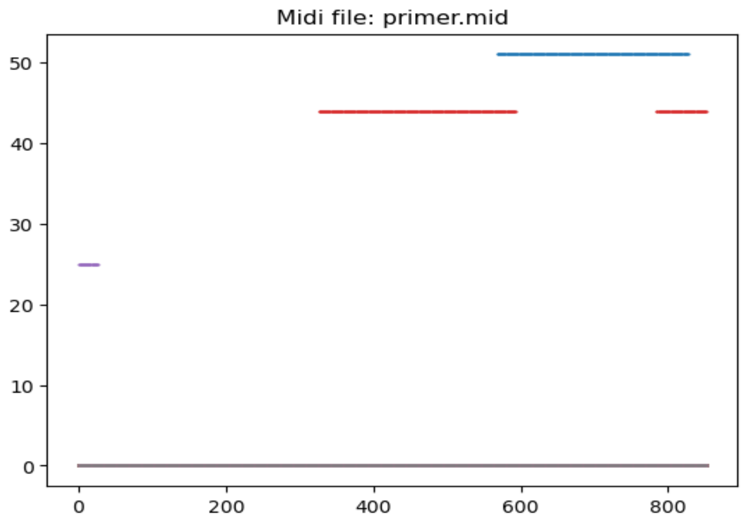


Figure 14: Retrained model Primer midi file
Notes v/s Time

A better plot showing the note against the time is Figure 8
A visually better plot that appears like in an equalizer is as shown in Figure 16

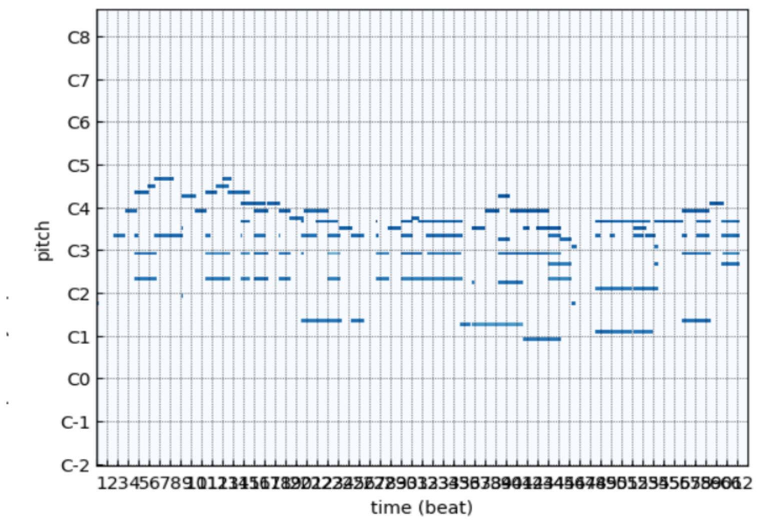


Figure 15: Retrained model Generated Music (pitch vs time)

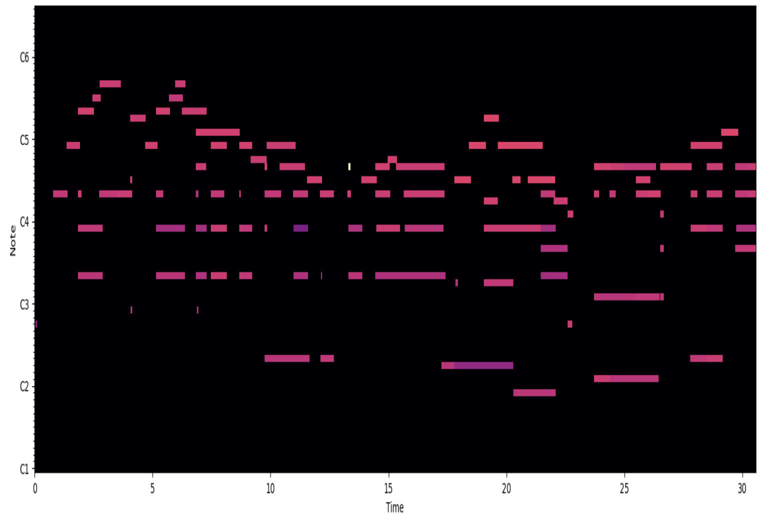


Figure 16: Retrained model Generated Music (notes vs time)

The music file generated with the retrained model is shown as a plot between notes versus time.

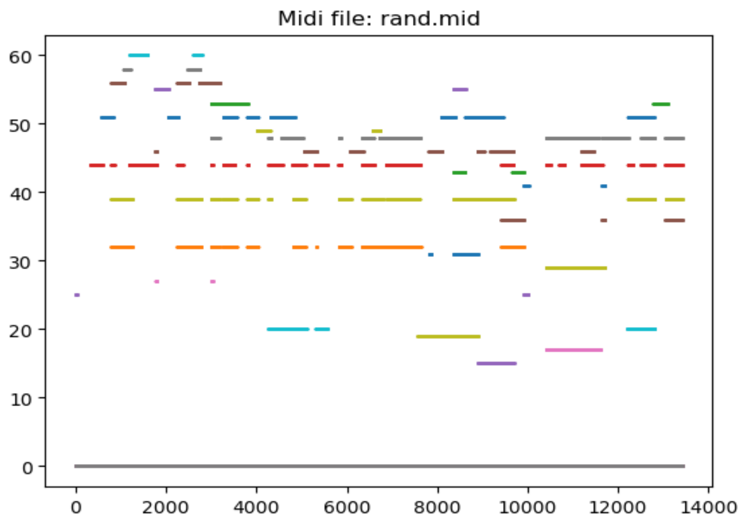


Figure 17: Retrained model Generated Music file

The outputs generated for each of the plot made here is available on [8].

9 Conclusion

Music which is a form of sound could be predicted by a model without the information about the frequency aspect of it. The loss obtained for the baseline model was 2.10 and is close to the results from the paper [3] Huang et al. For the retrained model, the loss was around 1.72 on Tegridy dataset.

10 Future Scope

The project was implemented to generate music from the notes of a piano. It would be interesting to see how the transformers could be implemented on different genres of music that have different instruments.

11 Challenges faced

The training time for training a baseline model for 150 epochs with maximum sequence length of 512 on a NVIDIA 1660 Ti GPU took around 12 hours to complete. On attempting to retrain the model for another 50 epochs on our custom dataset, the GPU VRAM was not sufficient. So the model was deployed on a GCP instance with an NVIDIA Tesla P4 8GB GPU that could resolve the issue for training.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need. In Advances in Neural Information Processing Systems," 2017, CoRR, abs/1706.03762.

[2] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani," Self-attention with relative position representations," arXiv preprint arXiv:1803.02155, 2018.

[3] Cheng-Zhi A. Huang et al. 2019. Music Transformer: Generating music with long-term structure. In Proc. Int. Conf. Learning Representations.

[4] Douglas Eck and Juergen Schmidhuber. Finding temporal structure in music: Blues improvisation with lstm recurrent networks. In Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on, pp. 747–756. IEEE, 2002.

[5] Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., & Salakhutdinov, R. (2019), Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. https://doi.org/10.18653/v1/p19-1285

[6] Github codebases: Transformer-XL.

[7] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, Llion Jones, Character-Level Language Modeling with Deeper Self-Attention, International Multidisciplinary Research Journal, 50–59, 2022.

[8] Test Dataset: Tegridy Dataset

[9] Baseline Dataset: Maestro V.2.0 Dataset

[10] Results: Drive link

[11] Video: YouTube link