Classical Music Generator

Kwok Lam Him (1155144407) and Kwok Chun Kiu (1155141911)

I. Abstract

This project attempts to create a music generator based on the given genre using miniature GPT-3. The miniature GPT-3 facilitates the music generator to accept a MIDI sequence and applies the self-attention algorithm to generate each subsequent note based on the given input, creating a connected and logical musical piece. With the music generator, this project explores the rationale of the masked self-attention procedure and serve as hands-on practice for programming machine learning models.

II. Background

This project corresponds to the later part of the course, which explains the mechanisms of machine learning and algorithmic composition. In the lectures, Magenta is used to construct RNNs (recurrent neural networks) for generating music. Instead, this project tries to generate better music using miniature GPT model. The music generator applies the self-attention algorithm to relate previous notes and the current note, so that the piece of music sounds connected and harmonious. This will be elaborated in Section III-E.

GPT-3 from OpenAI is a pre-trained natural language processing model, which can generate human-like text, simulate question-and-answer scenarios, and process language-based environments. OpenAI applies a similar idea to create MuseNet [1], which predicts the next token or word in a sequence in an audio file. MuseNet can generate music given a preferred composer, instruments or style, and an excerpt of a famous piece. This project implements the idea with the restriction to a certain style (classical) and instrument (piano) to reduce training time and model complexity. This project is based on the work of Elliot Koh, Sean Lim, Sidharth

Praveen, and Viet Pham [2] on jazz music generation, documentation from Keras [3] on building a miniature GPT, and the thesis [4] introducing the transformer model.

III. METHODOLOGIES

This section introduces the programming language, libraries and algorithms used in this project.

A. Programming Language

In this project, Python is the only programming language used. Renowned for its simplicity and large amount of library support, many AI (Artificial Intelligence) engineers use it to construct prototype of machine learning models. The machine learning library in this project is TensorFlow.

B. MIDI File Encoding

Classical piano music MIDI data are downloaded from [5], which contains the velocity, pitch, start time and end time information for each note. However, with this style of encoding, the model is unable to learn the relation of notes between different timestamps or in between the sequence. Therefore, the whole piece of music is encoded before training to generate more suitable data for the model to train on. A 2-D array of size (number of 32nd notes in the music × 128) is created and the MIDI is sampled for every 32nd note. (The 32nd note is chosen to optimize the size of the training data while preserving the music details.) When a MIDI note is 'on' at that timestamp, the array representing the timestamp will have a '1' on the index same as the MIDI note number, and otherwise '0'. Here, every timestamp in the piece is tokenised. The libraries used in this step are `pretty-midi` and `music21`.

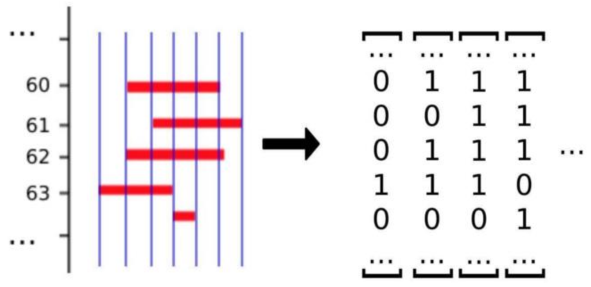


Fig. 1. An encoding example [2]

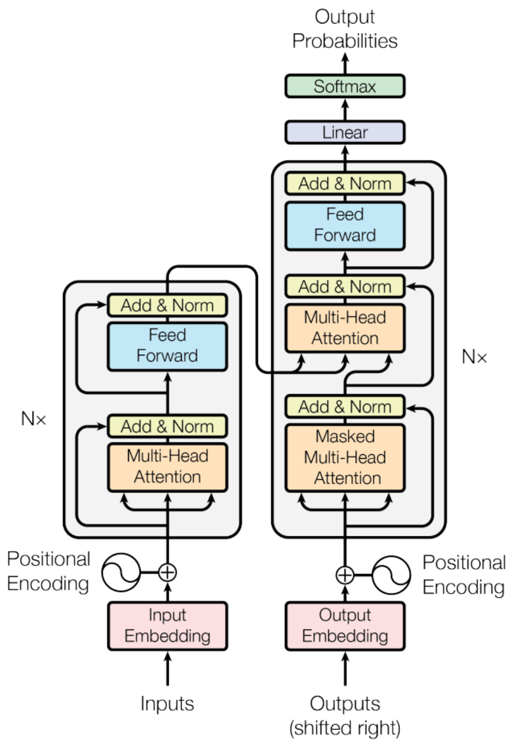


Fig. 2. Transformer model architecture [4]

C. *Embedding Layer*

Embedding layer is the first layer the inputs encountered. This is where the multi-dimension dictionary of the encoded array is constructed. The dimension is set to be 128 because it is the same as the range of MIDI notes number and is already the minimum dimension necessary to describe every combination of the elements in the tokenised timestamp.

D. *Positional Encoding*

Since this model has no recurrence or convolution, information about the relative or absolute position of the tokens must be added so that the model can learn the sequence order. There are many ways to encode the positional information, and in this model, sine and cosine functions of different frequencies are respectively implemented to even and odd positions.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Fig. 3. Sine and cosine function for positional encoding [4]

In Fig. 3, *pos* is the index of the position and *i* is the dimension of the model. The thesis [4] hypothesizes that "[the functions] would allow the model to easily learn to attend by relative positions, since for any fixed offset *k*, PE_{pos+k} can be represented as a linear function of PE_{pos}".

The outputs of both embedding layer and positional encoding are aggregated and passed to the self-attention algorithm as input.

E. Residual Connection

The transformer has residual connections, which is represented by arrows pointing at 'Add & Norm' layer. Therefore, the positional information can be leaked to the next layer and can propagate to further layers including the linear layer.

F. Self-Attention and Masked Self-Attention Algorithm

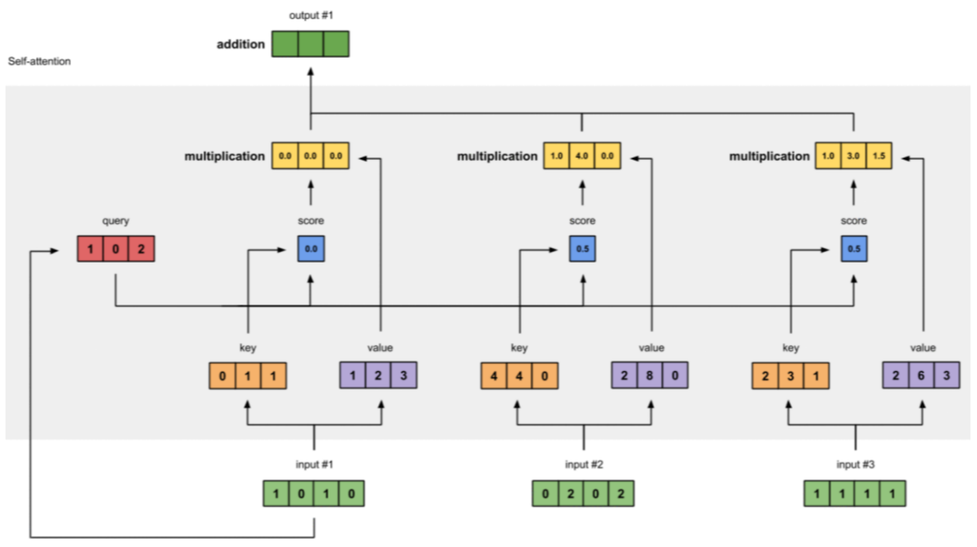


Fig. 4. Illustration of the self-attention layer [6]

Every input fed to the self-attention layer is converted to a value, a key, and a query by passing through the corresponding dense neural network layers. The 'key, value, query' concept can be thought of as searching for videos on YouTube. The search engine will map the query from one to a set of keys, including tags, keywords, and some other features. Then the best-matched candidate videos will be presented. In Fig. 4, to produce an output, there is a query from the input, and every key from all other inputs is combined (dot product) with the query to form a SoftMax score. For each score generated, it is multiplied by the value of its corresponding input. All the results from the multiplications are aggregated to determine the value of the output. To be more conceptual, all the previous inputs have different weights on calculating the output.

In the case of masked self-attention, only the keys from the inputs prior to the output position are included to the prediction during training to prevent the model from learning the future part of the output sequence that is solely for training and is not provided when predicting in normal usage.

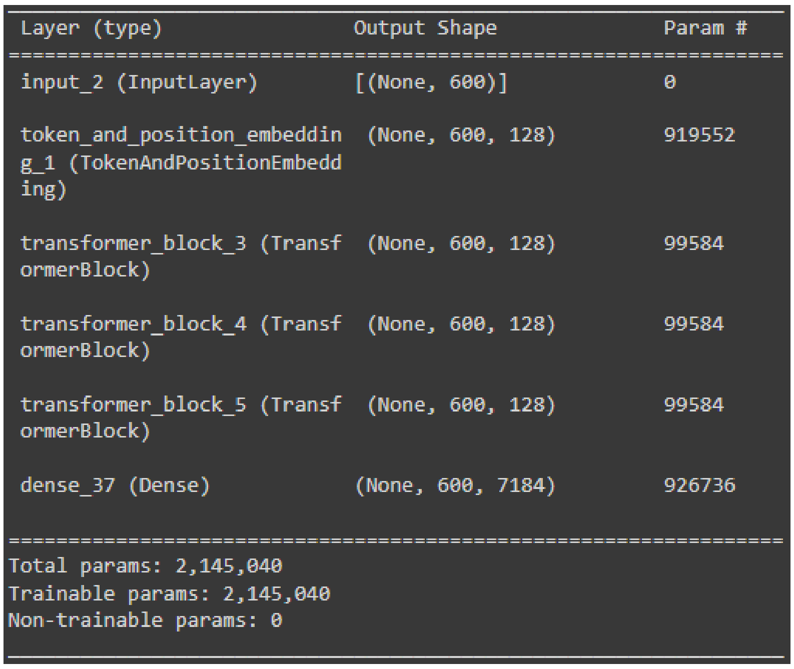
For the `TranformerBlock` used in this project, there is first a self-attention layer followed by one dropout layer, one normalization layer, one dense layer and then one dropout and normalization layer again. The dropout layer is to prevent the model from overfitting to the training data. The purpose of the normalization layer is to prevent the numbers in the network from growing too extreme. Together with the dense layer, a token-like vector can be outputted.

G. *Linear (Dense) Layer and SoftMax Activation Function*

After acquiring the value of the output, translation to an output vector is required so there is a densely connected layer with SoftMax activation function to determine the vector inside the dictionary with the highest probability of appearance.

H. *The model*

Fig. 5 shows the details of the model.



```
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 600)]             0

 token_and_position_embeddin  (None, 600, 128)         919552
 g_1 (TokenAndPositionEmbedd
 ing)

 transformer_block_3 (Transf  (None, 600, 128)         99584
 ormerBlock)

 transformer_block_4 (Transf  (None, 600, 128)         99584
 ormerBlock)

 transformer_block_5 (Transf  (None, 600, 128)         99584
 ormerBlock)

 dense_37 (Dense)            (None, 600, 7184)         926736

=================================================================
Total params: 2,145,040
Trainable params: 2,145,040
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5. The model used in this project

IV. CONCLUSION

In this project, we explored the masked attention algorithm and programmed the miniature GPT-3 model. The project provided the authors with a deeper understanding of the mechanisms of text vectorisation and allows further exploration towards natural language processing, the original purpose for GPT-3.

Both authors have programmed the pre-processing module and the GPT-3 model. The authors would like to thank Dr. Chuck-jee Chau for the advice on this project.

REFERENCES

[1] Payne, Christine. “MuseNet.” Open AI. https://openai.com/blog/musenet/ (accessed Dec. 20, 2021).

[2] Pham, Viet. “Jazz music generation using GPT.” Towards Data Science. https://towardsdatascience.com/jazz-music-generation-using-gpt-a7ed52b0f468 (accessed Dec. 20, 2021).

[3] Nandan, Apoorv. “Text generation with a miniature GPT.” Keras. https://keras.io/examples/generative/text_generation_with_miniature_gpt/ (accessed Dec. 20, 2021).

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, ... & I. Polosukhin. “Attention is all you need,” 2017. [Online]. Available: arXiv:1706.03762v5.

[5] Krueger, Bernd. “Classical Piano Midi Page.” http://www.piano-midi.de (accessed Dec. 20, 2021).

[6] Karim, Raimi. “Illustrated: Self-Attention.” Towards Data Science. https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a (accessed Dec. 20, 2021).