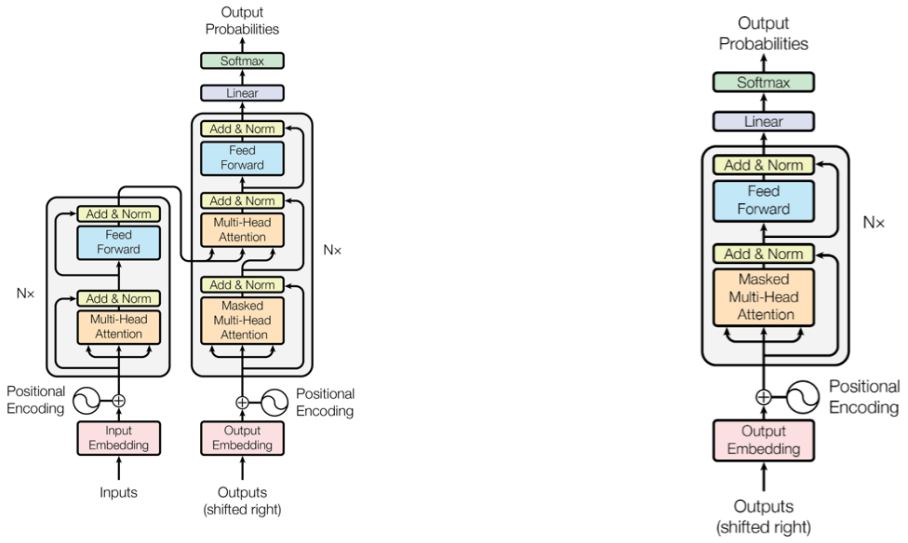
Error-Correcting Music Transformers

Jonathan Keane, Michael Conner, and Josiah Yoder
EECS
Milwaukee School of Engineering
Milwaukee, WI 53202
{keanej, connerm, yoder}@msoe.edu

March 18, 2023

Abstract

In state-of-the-art music Transformers today, a decoder-only Transformer autoregressively produces a musical work, basing its decisions for the next event on the previous sequence of events. With this process, however, there are no means for the model to iterate on itself and correct the original piece when musical inconsistencies occur. Additionally, because decoder-only Transformers only can look backward at the sequence created before it, current music Transformers have no future context that can provide the semantic meaning of future notes, which may be beneficial in informing better decisions about the correct note/event to be used at points earlier in the sequence. Therefore, we propose training a second encoder-decoder Transformer to correct music by training this Transformer to return songs with discretely inserted abnormalities back to its original piece. With this second Transformer, we can then use a generated piece of music from a decoder-only Transformer as the encoder input such that this "error-correction" Transformer can iterate on the original work to improve its quality. This encoder-decoder Transformer can then attend to the context from the whole generated piece and will have learned during training how to correct abnormalities it comes across.



(a) Encoder-Decoder Transformer (reproduced from [3])

(b) Decoder-Only Transformer (as in [1])

Figure 1: Two types of Transformers used for generating sequences.

1 Introduction

When musicians are composing a new piece of music, they do not often come across brilliance upon their first try. They often look at the piece they currently have and make small adjustments, such as making a note more staccato or legato to better fit the piece or changing the note that is played in a piece because it is too sharp or flat. While we see this in the case of human composers, state-of-the-art music Transformers today only perform a single pass while composing a song. Therefore, we hope to show with this research that by allowing a music Transformer to iterate and correct its original piece of music, we can improve the results of the current state-of-the art.

There are two basic architectures of Transformers used for generating sequences. The first kind of Transformer is the original model proposed by Vaswani et al. [3] (Fig. 1a) where there is both an encoder and a decoder, and the goal of the model is to be able to take a text from one "language" and generate a new sequence in a "target language." This encoder-decoder Transformer is commonly used in the NLP field for the task of translating between languages (for example, generating English translations from Spanish source-texts). The second form of the Transformer is a decoder-only model (Fig. 1b) where there is only an encoder block, which uses self-attention to extend an input sequence. This decoder-only Transformer is currently what is used in the state-of-the-art music Transformers (such as [1]), where novel musical works are being generated without a reference work from which they are derived.

While music Transformers [1] use the same general architecture as text-to-text translation Transformers, qualitatively, music generators do not yet achieve musical compositions with

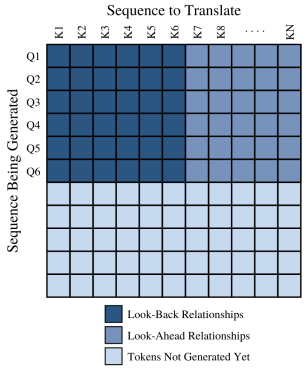


Figure 2: In autoregressively building a sequence with an encoder-decoder Transformer, when you perform cross-attention, the model can take context from information ahead and behind the token being generated.

the same level of coherence seen in text to text translations. One reason for this may be that single note events (such as a "note on" event) convey less information than a word of text. For example, in natural language, when you have the prefix "the dog" in a sentence, because a verb would follow this far more naturally than an adjective. For music, a single note could be followed by almost any other note on the scale, depending on what key the song is in. Thus, a music Transformer must model ideas such as key or phrasing in addition to the larger-scale "grammar" of music.

To address this challenge, we propose to improve the generative quality of music Transformers by training an "error-correction Transformer" that uses the encoder-decoder version of a Transformer (Fig. 1a) to translate music with errors ("bad music") into music without these errors ("good music"). We propose that by discretely augmenting a data set of professional music with errors inserted programmatically and training to correct these errors, this Transformer can learn to correct musical errors. With this, we can take this trained model to iterate on the music from a trained state-of-the-art music Transformer (based on the decoder-only model, Fig. 1b) that generates a sequence and improve upon the initial piece by correcting its "musical errors," improving the overall results of the model as a whole. By having a full song to provide context for sequence generation, when we perform cross-attention in our encoder-decoder Transformer during inference, we will have both look-back as well as look-ahead relationships in our attention matrix as seen in Figure 2. As a result, this model provides additional information to the decisions the model makes that are not possible in an encoder-only Transformer.

2 Prior Work

For the task of music generation, the music Transformer produced by Huang et al. [1] is the foundation that this work builds upon. In [1] Transformer using relative self-attention

described by Shaw et. al [2] was used and shown to be capable of producing music that captured long term structure, outperforming previously used techniques for music generation, including both an LSTM and a baseline Transformer with no relative self-attention, when evaluated by humans. This model was not rated as highly as the testing data produced by actual musicians. Our work attempts to improve the performance of the generated music toward the quality of actual musicians.

3 Experiments: Random Addition/Removal of Notes

With the proposed architecture combining both the encoder-decoder and decoder-only forms of the Transformer, we look to have an encoder-decoder Transformer that can learn to correct sequences of music that have randomly inserted/removed notes from a piece of music to the original piece from a professional musician. With inserting random augmentations into the a piece of music, we believe that if there are not enough augmentations, the Transformer will only learn to output the original piece because the model will have lower cost for keeping the errors than attempting to change them. In the case that there are many augmentations, the model may learn to ignore the encoder input, regarding it as random noise, learning only to heed the input prefix and its own previous output. We expect that there is some region that falls in between these two possibilities such that the model learns to correct to the original piece when it comes across these augmentations. We will perform a grid search on the number of insertions and deletions to try and find what balance of augmentations improves results.

To determine if there are improvements, we want to compare the results from the initial decoder-only Transformer to the results that are passed through the initial encoder-decoder Transformer. Users will be presented with the two samples for n songs and be asked to determine which they believe is the better music without knowing which Transformer produced which. This will then be used to determine the improvements in an A/B test. From these results, we will use statistical tests to see if our results achieve any form of statistical significance.

3.1 Decoder-Only Transformer & Data Set

Our decoder-only Transformer is the Music Transformer [1], a decoder-only model (Fig. 1b). We use an open-source implementation [1] as the basis for our model. We adjusted some of the hyperparameters and the model that we used for our experiments has the dimensions described in Table 1 above. This baseline model of the music Transformer was trained on the e-Piano competition data set. All pieces in this data set were solo pieces played on the piano. All pieces are of classical music, performed by expert musicians. For training our

[1] https://github.com/jason9693/MusicTransformer-tensorflow2.0

Table 1: Model dimensions of our baseline Transformer used for generating music.

Model Attribute	Dimension
Max Sequence Length	2048
Embedding Dimension	512
Heads per Attention Mechanism	8
Layers	6
Optimization	Adam[a]
Dropout	0.2

[a](learning rate = 0.001,$\beta_1 = 0.9$, $\beta_2 = 0.9$)

baseline model, we took the original piano-e-competition data set and added an augmentations on pitch (shifting the whole song up or down for pitch shifts of up to 3 notes) as well as time shift augmentations where we stretched/compressed the time shifts for an entire piece between notes by a factor of 0.05. At this point, our baseline Transformer makes sequences that are fairly well-formed pieces of music.

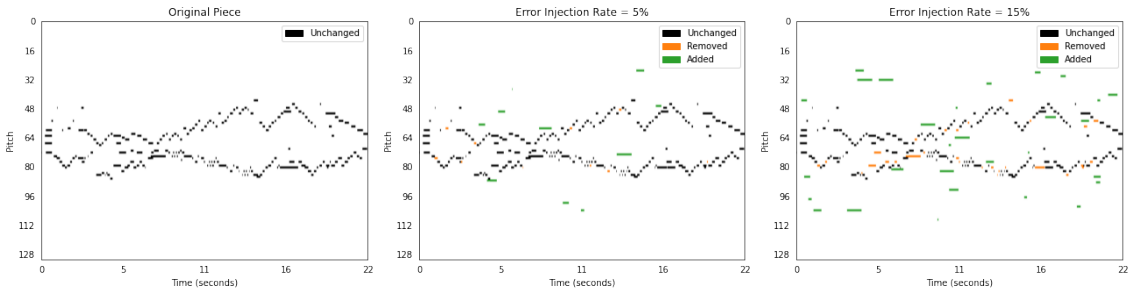
We encode performances as a series of events, using the encoding in Huang et al. [1]. This encoding has four different categories of events that represent events for NOTE_ON, NOTE_OFF, VELOCITY, and TIME_SHIFT. In the encoding scheme, there are 128 NOTE_ON and 128 NOTE_OFF events, which defines events to begin playing or stop playing a given pitch. 32 VELOCITY events are used to represent how hard any keys following this event will be pressed. Finally, to represent the natural timing of a performance, there are 100 TIME_SHIFT events that represent increases in the current timestamp in a piece, taking up all 10 ms intervals between 10 ms and 1 second. When a TIME_SHIFT event is processed, the current timestamp of the piece advances the defined amount and any notes that have been activated by a NOTE_ON event and have not been turned off by their respective NOTE_OFF event will be played for the duration specified by the TIME_SHIFT event.

3.2 Encoder-Decoder Transformer & Note Removal/Insertion

For the second Transformer being trained, we have an encoder-decoder setup that translates from one piece of music to another with differences. The dimensions of this model are defined in Table 1, but there is now both a decoder and an encoder in the model (as in Fig. 1a), which will be trained with encoder input as the augmented music and the expected decoder output as the original piece.

The augmentation algorithm we use randomly inserts and removes a specific number of notes. In our experiments, we took our samples from the beginning of the song, so that the error-filled input sequence and correct output training sequence would be more closely aligned. We do not expect that this would limit the trained model to operate only at the beginning of sequences.

Figure 3: Visualization of when notes are being pressed down during a song, comparing the original song, and the error-injected songs with error insertion rates of 5% and 15%, respectively. In the error injected figures, the notes removed from the original song and the new random notes added to the song are marked.



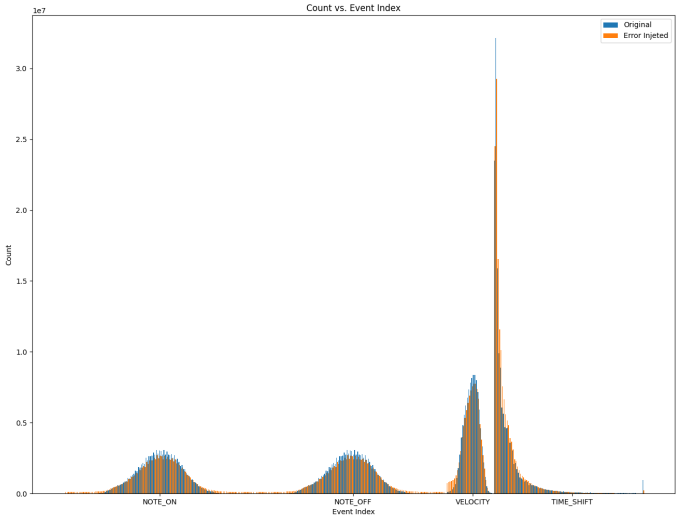
For our error injection procedure of adding/removing notes, in our preprocessing, for each sample in the augmented data set, we convert the event sequence to a list of full notes (encompassing the duration, pitch, and velocity) and use Algorithm 1 to remove and insert notes.

Algorithm 1 Error removal/insertion algorithm.

$P =$ number of pitches, $V =$ number of velocites, $T =$ number of time shifts
$R =$ removal rate
$N =$ notes
$U = Uniform([0,1))$
$D_{max} = \max\{\forall\ N_i\ \in\ N,\ Duration(N_i)\}$
$D_{min} = \min\{\forall\ N_i\ \in\ N,\ Duration(N_i)\}$
$L = \max\{\forall\ N_i\ \in\ N,\ StartTime(N_i)\}$
$V = U^{1\times|N|}$
$N_{errors} = \{N_i | V_i > R\}$
while $|N_{errors}| < |N|$ **do**
    $Start = U * L$
    $Duration = U * (D_{max} - D_{min}) + D_{min}$
    $Velocity = RandomInteger([0,V))$
    $Pitch = RandomInteger([0,P])$
    $N_{errors} = N_{errors} \cup \{Note(Start, Duration, Velocity, Pitch)\}$
end while
return N_{errors}

When we transform our original data set to uniformly distribute the notes/velocities/time shifts across their respective ranges, we see that the less frequent of each type of event in the original songs become more prominent in the error injected songs and vice versa for more common events in the original songs. This transformation of the distribution of events is visualized in Figure 4, as we see the NOTE_ON and NOTE_OFF events having their distributions become more uniform and flatten out across the entire range of possible notes. When we apply this transformation across the course of the whole song, we see that

Figure 4: Transformation of the distribution of the frequencies of the different note events across the original songs and their error-injected forms. The y-axis represents the number of each event across the training set.



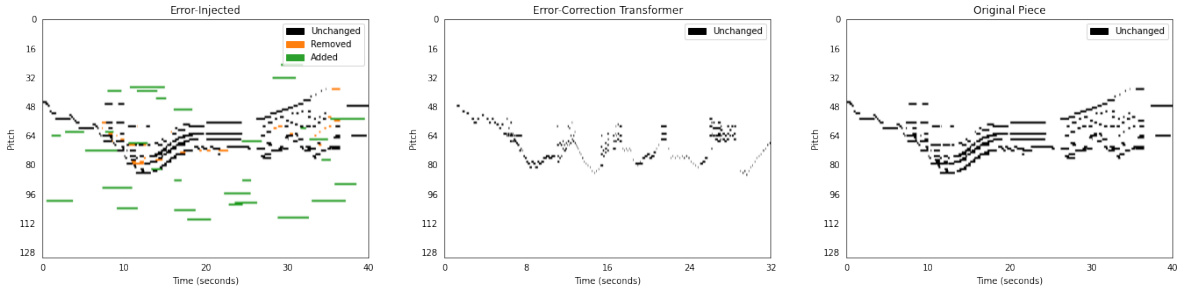
there is typically an increase in the average length of notes, as the correct distribution of the duration of notes is heavily weighted towards shorter notes compared to error-injected sequences across this data set, which is visualized in Figure 3, in the peak on the left end of the fourth rise.

3.3 Model Size & Training

We trained our Transformers on the ROSIE supercomputer at the Milwaukee School of Engineering on a DGX-1 pod using a single NVIDIA V100 Tensor Core GPU for 8 epochs across our augmented data set of 23072 samples, with each sample representing a whole song or an augmentation of a whole song. We use an 80/10/10 training/validation split of our data so we can observe the categorical accuracy measure on validation data during training, selecting the first 80% of the data set for the training set, the next 10% for validation, and the final 10% for testing to keep data consistently separated through different experiments.

3.4 Experiment Set 1: Batch Size = 1

In our first round of experiments, we used a batch size of 1 and a sequence length of 2048, to fit the model within memory during training. We tried training with error-insertion/removal rate from 2 to 30%, but the network continuously fell into producing repeated events or notes.



(a) (b) (c)

Figure 5: Example of the proposed error-correcting Transformer attempting to correct errors in a random error-injected song in the testing data. (a) Input to the encoder-decoder. Green bars show erroneous notes inserted into the song. Orange bars show notes removed (these are not given to the encoder-decoder). Black bars show notes retained. (b) Output of the encoder-decoder, generated with the goal of removing errors. (c) The original song corrupted to produce the input in (a).

3.5 Experiment Set 2: Batch Size = 8

Not finding any improvements with the hyperparameters explored in our first experiment set, we hypothesized that these failures could be due to a batch size that was too small to allow the model to generalize. We rationalized this as potentially being because with each training step, the model would gravitate towards the song it had just seen, but this would mean that the model would never have to try and synthesize results from many samples at once, leading to poor training results. To test this, we switched to a batch size of 8, while also reducing our max sequence length to 1024 to keep the entirety of our model within a single GPU. Performing training on the compressed vocabulary dataset with 15% error-insertion/removal rate, we saw that this model was able to exceed the accuracy score beyond what we saw in our first experiment set. In performing inference, generated sequences created coherent strings of notes.

This model could predict correct tokens in both the training and validation set with approximately 50% accuracy. While this measurement could potentially be misleading because the model could be learning to build sequences that used the most frequent events, a qualitative review of a small subset of the songs, such as the song demonstrated in Figure 5, showed that the pieces generated by the error-correction Transformer seemed to be much more authentically musical compared to the error-injected data it was given as input, while removing many of the errors injected into the piece. However, it seems that in learning to correct errors, the model learned to have a tendency towards shorter notes and less harmony when compared to the original piece. This may be because we inserted notes with random durations from the uniform distribution, our error-injected songs have longer notes compared to our correct songs, as we see the number of TIME_SHIFT events in Figure 4 has many more smaller TIME_SHIFT events compared to longer shifts. Therefore, the model may have learned to always use shorter notes because music with errors tended to

have longer notes, which is visible in the error-injected piece in Figure 5.

In a second experiment in this set, we began by training the Transformer to learn the identity transformation (encoder sequence producing a target sequence that was the same encoder sequence advanced one event) before switching to training with errors introduced in the input sequence. Even though we used a batch size of 8, this model trained poorly. We suspect that the pretraining may have hurt the performance by pruning out the more semantic relationships to favor weights that would be most suitable for the identity transformation only.

4 Conclusion

While we have not yet tested the encoder-decoder's ability to correct errors in music generated by a decoder-only music Transformer, in the preliminary experiments we present here, the music Transformer was able to both remove randomly-added notes and add in notes to compensate for randomly-removed notes, creating a coherent melody consistent with the corrupted source melody. With further experiments and tuning, this technique may be able to improve on the state-of-the-art in music Transformers.

5 Future Work

The encoder-decoder Transformer may have a more concrete understanding of what a note is because it must identify and remove spurious notes consisting of several related events: a velocity shift, a series of time shifts for the duration of the piece, and both a NOTE_ON and NOTE_OFF event. This potentially can be visualized by comparing the attention scores that come from the final head of Transformer when generating notes using the encoder-decoder Transformer compared to that of the original Transformer to see the attention to different previous notes. We would expect that in the encoder-decoder network, the weighting of the relationships of the NOTE_OFF event should have stronger weighting towards the NOTE_ON, VELOCITY, and TIME_SHIFT event that were part of the creation of this note, compared to these relationships in the decoder-only music Transformer.

Beyond improving unguided synthesis of music, the strategy proposed in this paper adds the value that we can inject knowledge from the problem domain into the network, making the model capable of being tuned towards any corrections that can be synthesized in the training output data and away from any errors that can be synthesized in the training input data.

There are a variety of strategies for synthesizing errors and improvements to a performance. Are there efficient ways to use errors synthesized by the generator Transformer? Could repetitions be inserted so the encoder-decoder learns to remove them? Can the melody line

of a performance be identified automatically and used to synthesize performance harmony from a performance of a novel melody? Error-correcting music Transformers have many possibilities.

References

[1] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, and Douglas Eck. An improved relative self-attention mechanism for transformer with application to music generation. *CoRR*, abs/1809.04281, 2018.

[2] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *CoRR*, abs/1803.02155, 2018.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.