Scientific Research Publishing

Art and Design Review, 2023, 11, 189-206
https://www.scirp.org/journal/adr
ISSN Online: 2332-2004
ISSN Print: 2332-1997

Coherent Music Composition with Efficient Deep Learning Architectures and Processes

Corey Zhang

Eastlake High School, Sammamish, WA, USA
Email: corzha.007@gmail.com

How to cite this paper: Zhang, C. (2023). Coherent Music Composition with Efficient Deep Learning Architectures and Processes. *Art and Design Review, 11,* 189-206.
https://doi.org/10.4236/adr.2023.113015

Received: April 28, 2023
Accepted: August 5, 2023
Published: August 8, 2023

Abstract

In recent years, significant advancements in music-generating deep learning models and neural networks have revolutionized the process of composing harmonically-sounding music. One notable innovation is the Music Transformer, a neural network that utilizes context generation and relationship tracking in sequential input. By leveraging transformer-based frameworks designed for handling sequential tasks and long-range functions, the Music Transformer captures self-reference through attention and excels at finding continuations of musical themes during training. This attention-based model offers the advantage of being easily trainable and capable of generating musical performances with long-term structure, as demonstrated by Google Brain's implementation. In this study, I will explore various instances and applications of the Music Transformer, highlighting its ability to efficiently generate symbolic musical structures. Additionally, I will delve into another state-of-the-art model called TonicNet, featuring a layered architecture combining GRU and self-attention mechanisms. TonicNet exhibits particular strength in generating music with enhanced long-term structure, as evidenced by its superior performance in both objective metrics and subjective evaluations. To further improve TonicNet, I will evaluate its performance using the same metrics and propose modifications to its hyperparameters, architecture, and dataset.

Keywords

Transformer, Attention, Long-Term Structure, Architecture

1. Introduction to the Music Transformer

Musical composition is the process of making or forming a piece of music by combining the parts, or elements of music by applying a system, structure, and motif throughout the entire piece. Most people come to music composition with a lifetime of listening experiences. Music composition takes learning one skill at a time and adopting it in the composition, whether it's harmony, melody, or form.

However, with the Music Transformer (Huang et al., 2019), we see a more innovative, dynamic, and progressive approach to music generation (Briot et al., 2019; Ji et al., 2020). Music transformers are revolutionary in the music industry as it enables automated music composition, allowing composers to generate original pieces by inputting a few parameters, such as genre, mood, or length, and the model can create fully orchestrated compositions. Before the introduction of transformers, recurrent neural networks (RNNs) with installed attention mechanisms were used in deep learning to develop models where output translations would be fed in as inputs, forming a recurrent process with temporal data, and making it proficient at language translation and processing. The transformer model, however, is able to match this performance without recurrent processing and limited training data with its cutting-edge operations with attention mechanisms. With the example of translation, words are rendered into vectors with different degrees and orientations. In a procedure of encoding and decoding the data, the model analyzes the source input, encodes and decodes it into weighted data, allowing it to predict the most accurate translation.

In general, the analysis of all AI-generated music software and models comes down to three important elements—hyperparameters, architecture, and dataset—which we will be evaluating in this paper. The concepts of hyperparameters, architecture, and dataset are closely interconnected when it comes to training and modifying a machine learning model like TonicNet. Making changes to these elements can ultimately help the model converge faster, avoid overfitting, or handle specific types of datasets, while improving feature extraction. Modifying hyperparameters can significantly impact the model's performance, convergence speed, and ability to generalize to new data, architecture refers to the overall arrangement and connectivity of its layers and the types of operations performed in each layer, while the dataset is essential for testing the model's performance through training and evaluating its results. Changes in the dataset can influence the model's ability to generalize, handle different types of inputs, and mitigate background noise present in the data, in the case that the input is in an audio file.

2. Data Processing with Music Transformer

2.1. Process

Starting by looking at the basic encoding protocol of RNNs commonly using MIDI files, we can see that a keystone concept of performance encoding is the serialization of micro-timing and velocity. This is derived from the event sequence of turning on and off notes, which captures the velocity and polyphonic structure of the dataset with global clock networks (GCLKs) specializing in this variety of digital signal processing. This includes a crucial step: quantization, in which the continuous timing information in the MIDI file is converted into discrete time steps that align with the beat and tempo of the music.

With the basic processing structure of Transformers, it is able to capture a more coherent and long-term structure and establish a dominant motif in the musical dataset. Specifically, after the audio is transcribed as symbolic music into a MIDI file and synthesized, the Transformer model is able to adhere to the theme that it captures, and essentially endures this theme throughout the predictive segment of the musical composition. Ultimately, it produces as output a sequence of new MIDI events that are intended to be a continuation of the input sequence. It does this by predicting the probability distribution over the set of possible MIDI events at each time step, based on the input sequence seen so far.

Sequence length normalization ensures that all input sequences have the same length, typically by truncating or padding them. This is achieved by dividing the data into subsets for training, validation, and evaluation. The preprocessing software for this process uses music21, a Python library for computer-aided musicology, providing a set of tools for working with music notation, analysis, and performance. It can be used to read and write music notation in various file formats, manipulate music scores and parts, perform music analysis, and generate new music compositions.

2.2. Self-Attention Mechanism

The Transformer employs the use of self-attention (Jagannathan et al., 2022), and this mechanism comes down to the layers of data from the encoder and decoder. The self-attention mechanism allows it to relate different positions of a sequenced encoder in order to compute a representation of the same sequence as a decoder. The implication of this procedure allows every position in the decoder to oversee all positions in the encoder sequence.

In essence, the process of self-attention, also known as multi-head attention in the applications of attention in the Transformers model, is the model's ability to procure values and queries from the previously processed layers of encoding. This produces the decoder's autoregressive modeling which enables it to manage all previously processed positions including the current position. Therefore, the self-attention technique implemented by the Transformer enables it to detect subtle ways in which data elements influence each other.

Relative Positional Self-Attention Shaw et al. (Shaw et al., 2018) introduces a methodical approach: using position encodings with input elements to expose position information to the model, they can be a deterministic function of position or learned representations. It is an extension of the Music Transformer self attention mechanism, which allows the model to weigh the importance of different parts of the input sequence when making predictions. Convolutional neural networks inherently capture relative positions within the convolution window. They proposed an extension to self-attention to consider the pairwise relationships between input elements by modeling the input as a labeled, directed, fully connected graph and modified the Transformer's self-attention mechanism to consider the relative distances between sequence elements. The authors proposed adding a set of trainable embeddings to the Transformer so as to make its output representations also represent the sequential information in its inputs. These vector embeddings are employed in the process of computing the attention weight and value between two words in the input sequence. These embeddings represent the distance (number of words) between words; thus, the name Relative Position Representation (RPR) was coined.

Relative Positional Self-Attention takes into account the relative positions of the elements in the input sequence, as well as their absolute positions. This is important in musical contexts, where the timing and rhythm of notes are crucial. By incorporating relative positional information, the model is better able to capture the structure and relationships within the input sequence.

In practice, Relative Positional Self-Attention is implemented using learned position embeddings, which are added to the input embeddings before being fed into the attention mechanism. The position embeddings capture information about the relative positions of the elements in the sequence, and are learned during training along with the other parameters of the model.

2.3. Architecture

The Music Transformer architecture consists of an encoder and a decoder, similar to the original Transformer architecture (Hsu & Chang, 2021). The encoder processes the input sequence and generates a set of "context vectors" that are used by the decoder to generate the output sequence. The decoder is responsible for generating the output sequence, one event at a time, based on the context vectors and the events generated so far. The Music Transformer can be trained on a large corpus of MIDI files using maximum likelihood estimation, where the goal is to maximize the probability of generating the correct output sequence given the input sequence. Once trained, the model can be used to generate new music by sampling from the output distribution at each time step, or by using beam search to find the most likely sequence of events.

2.4. LSTM

LSTM (Long Short-Term Memory) is a type of RNN architecture that was developed to address the vanishing gradient problem in traditional RNNs.

In traditional RNNs, the gradient signal can become smaller and smaller as it is propagated back through time, which can result in the weights of the network not being updated effectively and the network being unable to learn long-term dependencies. LSTM networks solve this problem by introducing a memory cell that can store information for long periods of time, along with three gates (the input gate, forget gate, and output gate) that regulate the flow of information into and out of the cell.

The input gate controls whether to allow new information to enter the memory cell, the forget gate controls whether to erase old information from the memory cell, and the output gate controls whether to output information from the memory cell to the next layer of the network. By using these gates to selectively update the memory cell, LSTM networks can effectively learn long-term dependencies and are particularly well-suited for tasks that involve sequential data, such as natural language processing and speech recognition.

Encoder. To analyze the encoder of the Transformer, we can take language translation as an example. In contrast to the slow-processing RNNs, the Transformer can simultaneously embed all the inputs without loss of gradients, because the input sequence of the sentence is passed in parallel. In the Music Transformer, a sequence of musical events is taken as input takes as input and represented as one-hot vectors. They are mapped to a sequence of continuous representations, or embeddings, that capture the underlying structure of the music. Each embedding is computed by applying a series of multi-head self-attention layers and position-wise feedforward layers to the previous embeddings in the sequence. An important component of this is the embedding space, where the word is assigned to a vector. In positional encoding, the vector is assigned that provides context in regard to its position in the dataset. An essential part of this process is the multi-head attention mechanism whereby the inputs are run in parallel and then output a concatenated representation (Figure 1).

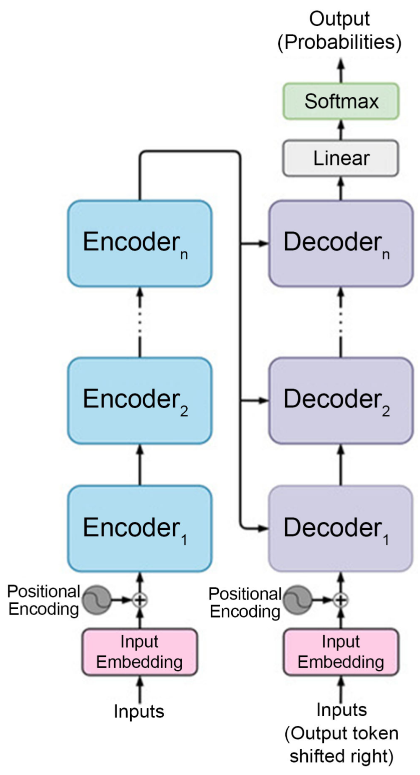

Figure 1. Transformer's encoder-decoder architecture.

Decoder. During the training phase in language translation, in order for the computer to undergo the translation phase, the model must be fed information that it can decode. In this case, it would be the input embedding from the encoder and also the positional vector in order for it to understand the structure of the sentence and the context of each word. These vectors relate each word in the sentence to every other word in the sentence, and as each word is fed into the attention mechanism, feedforward layers make the output easily computable to the computer, allowing it to predict the next word that logically follows. Similarly, in the Music Transformer, a sequence of embeddings generated by the encoder is taken and it autoregressive generates a sequence of musical events. At each time step, the decoder takes as input the embedding of the previously generated event, as well as the context vectors computed by the multi-head self-attention mechanism over the encoder output. It then applies a series of self-attention layers and position-wise feedforward layers to the previous embeddings to generate the embedding of the next event, which is then decoded into a musical symbol using a softmax classifier.

3. Other Methods

We will first look at some techniques, architectures, and elements crucial to model the complex relationships between musical notes and rhythms in a way that can capture the flow and structure of music. These methods can play a large role in the generation of new sequences of music by sampling from the probability distribution of the model, allowing for the creation of novel and unique musical compositions.

Dropout. Srivastava et al. (Srivastava et al., 2014) introduced the fundamental application of dropout, a regularization technique for neural networks, drops units and connections at specified training times with specific probabilities, and "each hidden unit in a neural network trained with dropout must learn to work with a randomly chosen sample of other units. This should make each hidden unit more robust and drive it towards creating useful features on its own without relying on other hidden units to correct its mistakes". Srivastava et al. hypothesize that by nullifying the presence of hidden layers, dropout prevents co-adaptation, or a correlation of structural composition of each hidden unit. While all units are present, weights are scaled by probability. This mechanism prevents a situation where the neural network becomes too dependent on specific connections, as this could be an indication of overfitting and becoming too similar to the training data. Therefore, we can consider dropout as the creation of an implicit ensemble of neural networks. In many Music Transformer-based models, dropout can be used to randomly drop out some of the connections between the neurons in the network during training. This helps to prevent the network from relying too heavily on any one feature or set of features, and encourages it to learn more rich representations of the processing data (**Figure 2**).

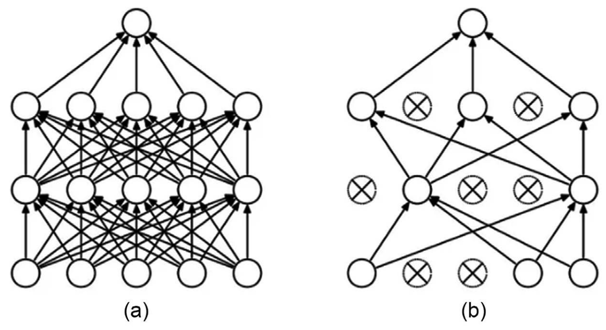

Figure 2. The use of regularization methods to drop nodes in the input and hidden layer to address overfitting. (a) Standard neural net; (b) After applying dropout.

Gated Recurrent Unit. A Gated Recurrent Unit (GRU) is a type of recurrent neural network very much similar to a long short-term memory (LSTM) (Dua et al., 2020). However, it only has two gates: a reset gate and an update gate. GRUs have fewer parameters, as they lack an output gate, thus making them generally easier to train than LSTMs. It is often employed in place of the original transformer layers. The GRU for Transformer was proposed as a way to reduce the computational cost of the original Music Transformer while still achieving good performance on music generation tasks. The use of GRUs allows the model to learn and remember longer-term dependencies in the music, which is important for generating coherent and musically pleasing sequences.

Softmax. The softmax output function takes the output of a previous layer and converts it into a vector of probabilities. In the output, the final layer of the neural network will be represented as a probability distribution over possible next notes or chords in a musical sequence. This is then used to sample the next note or chord to be added to the sequence. The softmax function ensures that the output probabilities sum to one and that the output values are between 0 and 1, allowing the model to generate a valid probability distribution over possible notes or chords. In the context of multiclass classification, an input vector and a weighting vector can be expressed with:

$$P(y = j|x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T w_k}}$$

Variational Dropout. Variational Dropout is described as a regularization technique constructed on the fundamental concepts of dropout techniques but employs variational inference in its approach. In Variational Dropout, a dropout mask with each element of the mask is a Bernoulli random variable that takes a value of 1 with probability p, and a value of 0 with probability 1 - p, where p is the dropout rate is cycled through at each time step for inputs, outputs, and recurrent layers while dropping the same network units during its traversal. This is essentially different from the regular Dropout data for faster processing speed, as various dropout masks are sampled at each individual time step solely to retrieve the inputs and outputs individually.

4. Related Experiments

4.1. TonicNet

Perecha (Peracha, 2019) has developed TonicNet in an effort to focus on validation set loss of the Transformer rather than architecture in his refinement model. TonicNet is a GRU-based model trained to initially predict the chord at a given time-step before predicting the notes of each voice at that time step, contrasting with the typical approach of predicting only the notes, while generating melodies and chord progressions. It is based on a recurrent neural network architecture and uses a combination of self-attention and relative position representations to capture the long-term dependencies in music. The model can either be trained to predict preconditioned features as well as the extra components of the sequences being modeled, significantly improving the overall results when we analyze the model performance. The goal of TonicNet is to create music that sounds more natural and coherent, and it has been trained on a large dataset of MIDI files from various genres.

Process and Architecture

RNN. TonicNet uses Recurrent Neural Networks (RNNs) to learn and generate music. RNNs are a type of neural network that has a feedback mechanism that allows information to be passed from one step of the network to the next. In TonicNet, the RNN is trained on a large corpus of music, and learns to predict the next note in a sequence given the previous notes (Figure 3).

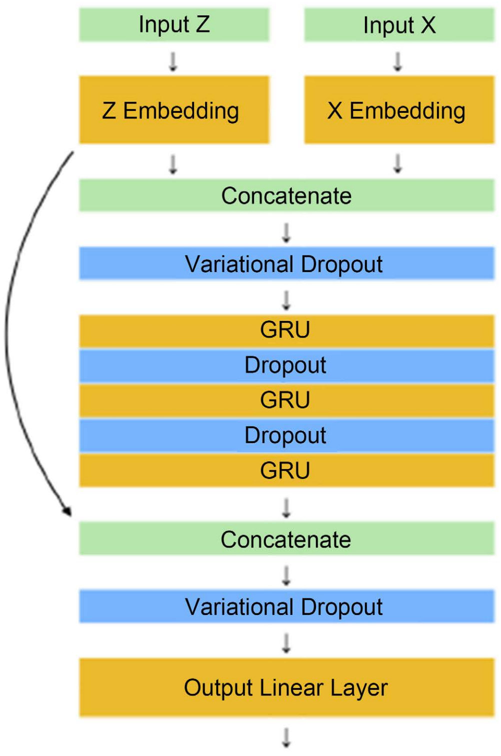

Figure 3. TonicNet's GRU architecture for time-step prediction.

This process is called sequence modeling. TonicNet uses a specific type of RNN called a Gated Recurrent Unit (GRU), which is designed to address the problem of vanishing gradients in traditional RNNs. The GRU allows information to be selectively passed on from one time step to the next, while filtering out irrelevant information. During the training process, TonicNet learns to generate music that is similar to the input corpus. The output music can be controlled by adjusting various hyperparameters, such as the number of RNN units and the length of the generated sequence.

TonicNet has two inputs: an integer value represented by x storing the class label for the previous time-step's class label, while the second integer represented by z stores the repetition count. These inputs are each converted to vector representations and by an embedding respectively. The embedding outputs are then concatenated with both the class label embedding, and the repetition count embedding processed during training from scratch. With the parameters that are used by Peracha, the concatenated embeddings have a Variational Dropout function applied with a 0.1 rate. This is then processed to a 256-unit GRU with three layers alternating with two layers of dropout applied after each of the first two layers at a 0.3 rate.

This process is complemented with a 5-layer Transformer encoder model. Layers refer to the individual building blocks that make up the neural network. A layer can be thought of as a computational unit that performs a specific operation on its input data and produces an output. Different types of layers in a neural network can include convolutional layers, activation layers, pooling layers, and fully connected layers. The number and types of layers used in TonicNet can greatly impact its performance and ability to learn from data. Perecha encodes an absolute position in the time-step sequence through a position embedding with 256 dimensions, which is processed by the model as an input embedding. A feedforward layer with 1024 units and set with a dropout rate of 0.1 within the encoder module has 1024 units has the input masked to ensure the model can only rely on the sequence's previous time-steps when making a prediction. Unlike other models, repetition encoding is not used whatsoever when training this model.

4.2. Quantitative and Qualitative Evaluation

We will be assessing the performance of AI models in generating music using both objective and subjective measures. The quantitative evaluation of ten involves measuring the model's accuracy, precision, recall, F1 score, and other performance metrics using established benchmarks or ground truth data. The qualitative evaluation often involves assessing the generated music's quality, creativity, coherence, and aesthetics, among other subjective measures, through surveys, human expert evaluations, and other means (**Figure 4**).

NLL. The NLL (negative log likelihood) loss is our considered benchmark in our qualitative evaluations of the models. When we look at this metric generally,

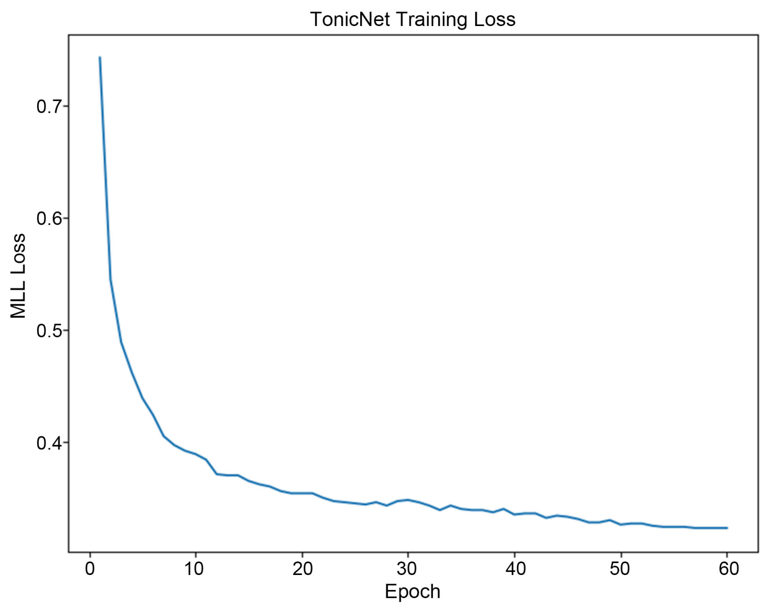

Figure 4. TonicNet NLL loss training data.

relative attention achieves a better NLL loss as compared to vanilla Transformers and other architectures. The benchmark is calculated when taking the log of the probability value after the softmax function has been applied and subsequently adding the probability value of the result to the average. In the music modeling sector, we see TonicNet currently having the best NLL metric as measured at 0.220. In Perecha's training from scratch based on the 60-epoch run, we see an optimal performance of a loss of 0.317. In my personal replications running with the 60-epoch run, I saw a 0.323 loss of optimal performance using the same hyperparameters and training from scratch.

In the context of machine learning and deep learning, NLL loss is a type of loss function that is used to measure the error or difference between the predicted output of a model and the actual target output. It is commonly used in multi-class classification problems, where the goal is to assign an input data point to one of several classes.

The NLL loss function is based on the concept of maximizing the likelihood of the correct class label given the input data. The negative log-likelihood is taken of this probability, which is then used as a measure of the error. The idea is to minimize this error during training by adjusting the model's parameters using techniques like gradient descent.

TonicNet was evaluated on the JSB Chorales dataset using the NLL loss and achieved a test set NLL of 0.88 bits per note. This was compared to the previous state-of-the-art model, which achieved a test set NLL of 0.94 bits per note and a Tonic Identification Accuracy (TIA) score of 91.8% compared to the sub-90% losses of previous models. These results demonstrate the effectiveness of TonicNet in modeling tonal music and generating coherent and tonally consistent musical sequences (**Figure 5**).

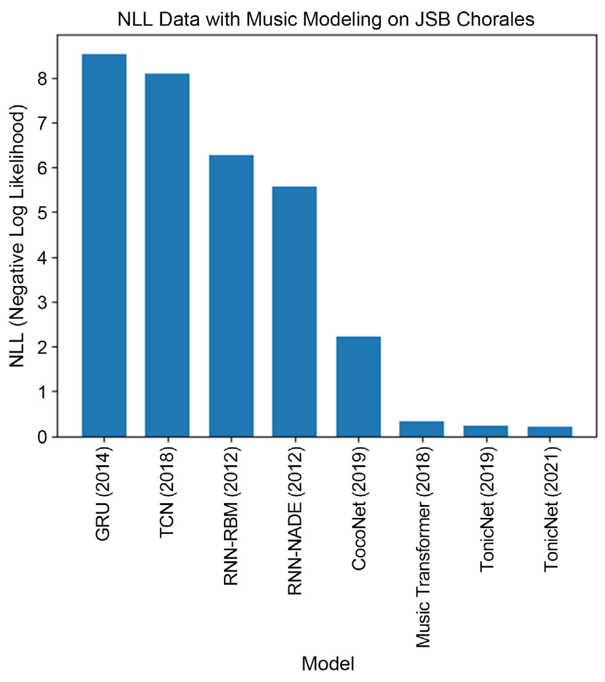

Figure 5. NLL benchmark statistical comparisons between different standard transformer models.

TonicNet has shown promising performance in generating music with tonal coherence. According to the paper by Peracha, it achieved over an 85% success rate with correctly predicted tonics (over 85%) in the test set, as well as over a 95% success rate for high mean pitch accuracy in the generated melodies. Additionally, the generated music was evaluated by human listeners in a listening test, where it was found that the generated music was often perceived as coherent and musical.

For the Vanilla Transformer, it is common for it to be evaluated against the metrics of accuracy, perplexity, and F1 score. For language modeling tasks, lower perplexity indicates better performance, while for classification tasks, higher accuracy and F1 score indicate better performance. The original transformer paper by Vaswani et al. (Vaswani et al., 2017) reported state-of-the-art results on the machine translation task using the WMT 2014 English-German dataset, achieving a BLEU score of 28.4 (**Figure 6**).

5. Refinements and Experimentation

5.1. Potential Improvements

First, TonicNet was trained on a specific dataset, and its performance may be limited if applied to a different musical style or genre. This makes TonicNet susceptible to overfitting the training data. This risk can be mitigated by using techniques such as dropout and early stopping.

As with many deep learning models, it can additionally be difficult to interpret how TonicNet arrives at its predictions. This may limit its usefulness in certain contexts where interpretability is important. While TonicNet may perform

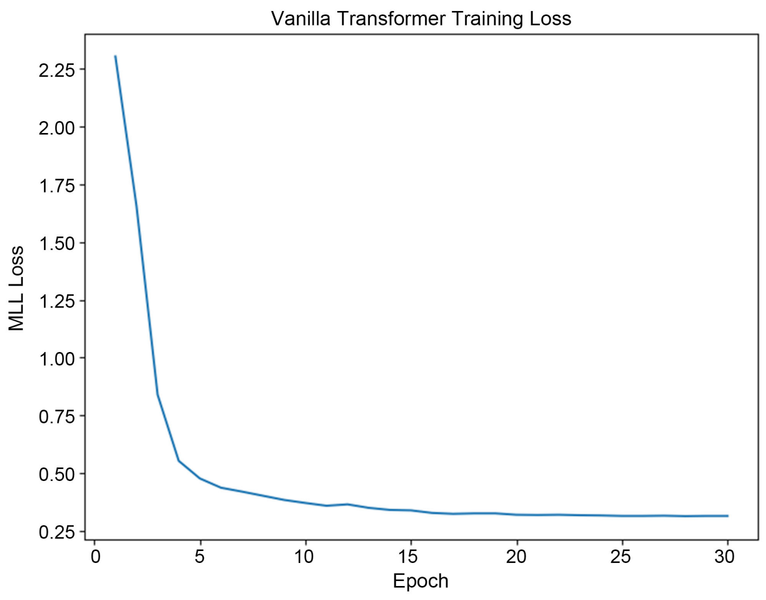

Figure 6. Vanilla transformer NLL loss training data.

well on generating novel music within the range of its training data, it may not necessarily capture the essence of the musical style or be able to generate music that is considered high quality or stimulating.

Refinements. In an effort to improve and optimize the qualitative performance of TonicNet, I will be tuning several of TonicNet hyperparameters, focusing on the RNN units and dropout rate. TonicNet is a neural network architecture used for audio classification tasks, and it includes a recurrent neural network (RNN) to model the temporal dependencies of the audio data. The number of RNN units, also known as hidden units, is a hyperparameter that controls the capacity and complexity of the RNN.

Changing the number of RNN units in TonicNet can have several effects on the network's performance and behavior, including the capacity of the network; a larger capacity allows it to capture more complex temporal patterns in the audio data improving the accuracy of the model, especially if the input data contains long-term dependencies. Nonetheless, increasing the number of RNN units could potentially lead to overfitting if the training data is not sufficient. If the number of RNN units is too small, the network may not be able to capture important temporal patterns in the data, and this will be reflected in poor accuracy results. In this case, increasing the number of RNN units can improve performance. Generally, we can approach this problem by starting with a moderate number of RNN units and gradually increasing or decreasing it while monitoring the performance on a validation set.

The implementation of TonicNet's dropout regularization to prevent overfitting is a technique that randomly drops out (sets to zero) some neurons during training to force the network to learn more robust features. Changing the dropout rate in TonicNet can have several effects on the network's performance and behavior. Higher dropout rates can increase the network's generalization ability by reducing overfitting. However, too high a dropout rate may lead to underfitting, where the network fails to learn important features. Lower dropout rates may result in better accuracy on the training data but can also lead to overfitting. This means that the network may perform well on the training data but generalize poorly to new, unseen data. The dropout rate can affect the convergence speed of the network. Too high or too low a dropout rate may slow down the training process. Thus, with the dropout rate, I will conduct careful experimentation and tuning to find the optimal value that balances preventing overfitting, improving generalization, and maintaining convergence speed.

5.2. Quantitative Results

In order to achieve the optimal coherence of the music patterns, we will first need to adjust hyperparameters to reduce the NLL loss in order to ensure that the network of the model is capable of predicting the correct notes and timing in the generated music. Therefore, reducing the NLL can help improve the overall quality of the generated music by making it more accurate and closer to the training data. We will not need to adjust the number of layers in the GRU architecture as the preset 3 layers is the optimal standard for the number of hidden layers and passing this threshold might result in an increase in time complexity concerning accuracy gain. We will first tweak the nb_rnn_units in TonicNet, which refers to the number of units (or neurons) in the recurrent neural network (RNN) layer of the model. The RNN layer is responsible for processing sequential data, such as time-series or text data, by maintaining a hidden state that captures information from previous time steps.

The number of units in the RNN layer determines the complexity of the model's representations and its ability to capture long-term dependencies in the input data. Typically, increasing the number of units can improve the model's performance but also increases the computational cost and risk of overfitting. Increasing the number of nb_rnn_units in TonicNet generally improves its ability to model and generate complex musical patterns and structures. This is because the RNN units in TonicNet learn to capture patterns in the input data over time, and increasing the number of units can provide more capacity for the model to learn complex patterns and relationships. We will be attempting the common approach of slowly incrementing the RNN units and seeing how this affects the overall loss function that is plotted as the number of units increases.

Increasing the number of RNN units in TonicNet can initially improve the model's loss because it allows the network to capture more complex temporal patterns and dependencies in the input music data. With more RNN units, the model can potentially better learn the underlying structure of the music and generate more coherent and musically pleasing sequences. However, increasing the number of RNN units beyond a certain point may lead to overfitting, where the model performs well on the training data but poorly on new, unseen data. As the capacity of the model increases, it becomes more prone to overfitting, meaning that it begins to memorize the training data rather than learning general patterns. This results in a decrease in the model's ability to generalize to new, unseen data, and a corresponding increase in its loss on the validation set. As we can see, testing the incrementation of the number of RNN units in 20-unit increments, we find that In conclusion, it is important to strike a balance between model capacity and generalization by choosing an appropriate value for the nb_rnn_units parameter and possibly adjusting other hyperparameters, such as dropout, to prevent overfitting. Thus, I will then adjust the dropout rate to prevent poor generalization to new, unseen data. Essentially, we want to reduce the complexity of the model and evade the situation where it starts to memorize the training data instead of learning more general patterns that can be applied to new data.

To address overfitting, I will try reducing the number of layers or adding regularization techniques such as dropout, weight decay, or early stopping. These methods can help the model to generalize better to new data and prevent overfitting. Although I can try increasing the size of the training dataset, since having more data can help the model to learn more general patterns and avoid overfitting, we can also tweak the dropout rate to adjust the proportion of neurons that will be dropped during training.

Increasing the dropout rate in TonicNet can have both positive and negative effects on the network's performance, depending on the specific task and dataset. Here are some potential effects. Dropout is a regularization technique that helps prevent overfitting, which occurs when a model fits too closely to the training data and fails to generalize well to new, unseen data. Increasing the dropout rate can improve the model's generalization performance by reducing overfitting, especially if the model is complex and has many parameters. Dropout can also be seen as a form of noise reduction because it encourages the network to learn more robust and invariant features that are less affected by noise and variations in the input data. In general terms, this can be especially beneficial if the input audio data contains a lot of background noise or other variations in sound. Increasing many of the hyperparameters such as the dropout rate can result in a reduced capacity of the model, making it harder for the network to learn important patterns in the data. I saw that this resulted in underfitting, where the model is not able to capture the complexity of the audio data and performs poorly on both the training sets (**Figure 7**).

In my experimentation with the dropout rate, I found that increasing the dropout rate can slow down the convergence speed of the network because more training iterations are required to compensate and strive for an on-par level of accuracy, increasing the training time and computational cost if the dataset or the model is complex. Overall, increasing the dropout rate in TonicNet requires careful experimentation and tuning to find the optimal value that balances preventing overfitting, improving generalization, and maintaining the network's capacity and convergence speed.

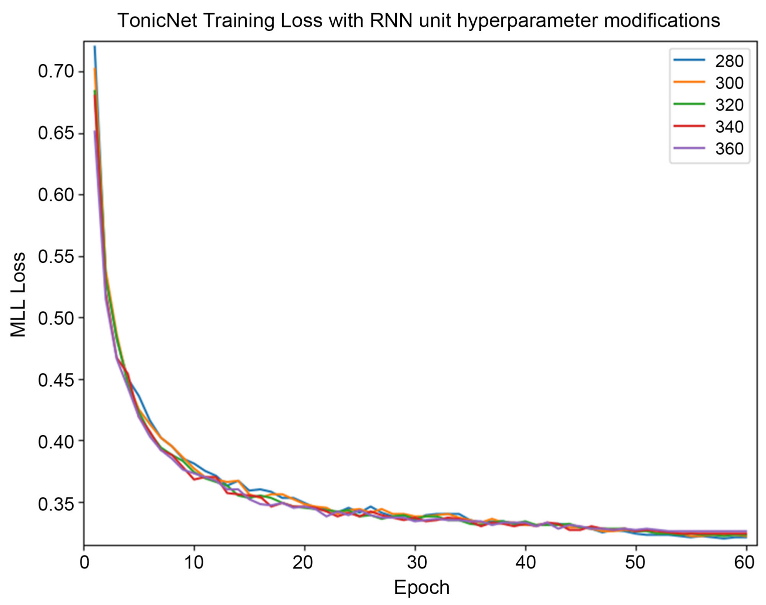

Figure 7. TonicNet NLL loss training data with RNN and hyperparameter modifications.

I slowly tested out the dropout rate by incrementing the dropout rate after each testing phase to address issues with overfitting and identifying improvements in generalization performance of the model. However, I must be careful not to set the dropout rate too high, and cause the network to lose important information during training, making it harder to learn useful features. I saw this setting the dropout rate to 0.5, as I speculate that too many neurons were dropped out during training, leading to an evident overall decrease in performance. A lower dropout rate may be more appropriate for the specific dataset and model architecture and we also must note that the optimal dropout rate may vary depending on the specific task and dataset being used. As seen with the results, the optimal dropout is seen at 0.4, as trying to decrement the dropout to 0.2 also caused a poor generalization of the validation set and also resulted in overfitting.

5.3. Qualitative Results

TonicNet's quantitative performance can be evaluated using metrics such as Mean Opinion Score (MOS), root mean square error (RMSE), and signal-to-noise ratio (SNR) (Bhagat, Bhatt, & Kosta, 2012). MOS is a subjective measure that can be obtained from human listeners who rate the quality of the generated audio. RMSE measures the average difference between the predicted audio signal and the ground truth audio signal. SNR measures the ratio of the power of the audio signal to the power of the background noise. A lower RMSE and a higher SNR indicate better performance of the TonicNet model (**Figure 8**).

Increasing the number of RNN units in TonicNet may also improve the sound quality by allowing the model to capture more complex patterns in the input data, and therefore generate more accurate and diverse output sequences. In my

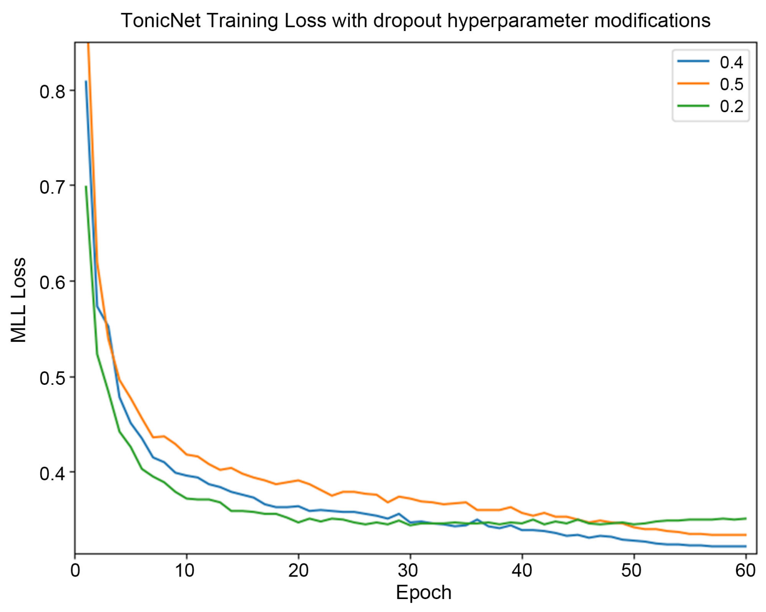

Figure 8. TonicNet NLL loss training data with dropout hyperparameter modifications.

qualitative evaluation of the generated datasets with the desired hyperparameters, I will listen to the generated music and assess its overall quality, including how smooth and clear the melody is, if it has variations and dynamics, while also assessing the coherence of the generated music to evaluate if it sounds disjointed. I will then compare this performance to other models. In many surveys, TonicNet's qualitative performance is generally considered to be good, with many samplers reporting that the generated music sounds pleasant and coherent (Chu et al., 2022).

My adjustments with the RNN units and individual analysis of the generated mp4 datasets definitely showed an increased performance. I found a more efficient attempt at capturing more variation and creativity in the music, making the generated result sound more human-composed. The melody also introduced a smoother flow with better incorporation of melody.

6. Conclusion

The importance of music transformers lies in their ability to model long-term dependencies in music sequences and generate coherent, high-quality musical compositions. Traditional recurrent neural networks (RNNs) struggle with modeling long-term dependencies, as the gradients can vanish or explode during training. Transformers overcome this limitation by using a self-attention mechanism and architecture that allows them to selectively attend to different parts of the input sequence, making it one of the most flexible, dynamic, scalable, and most importantly, coherent models in the recent years of developments in deep learning music composition. TonicNet's musically coherent and structurally robust architecture makes it a model for very high-quality AI music generation, especially when we consider the creativity and flexibility in music generation. My combined quantitative and qualitative evaluation by adjusting very critical hyperparameters, the number of RNN units and the dropout, helps determine the model's strengths and weaknesses, identify areas for improvement, and provide insights for future research and development, in order to address its especially limited output variety and attempt to tackle or at least mitigate the effects of overfitting. Through further training on diverse and representative datasets, we can then take steps to achieve an indistinguishable AI music generation sound from human-composed music.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

Bhagat, D., Bhatt, N., & Kosta, Y. (2012). Adaptive Multi-Rate Wideband Speech Codec Based on CELP Algorithm: Architectural Study, Implementation & Performance Analysis. In *2012 International Conference on Communication Systems and Network Technologies* (pp. 547-551). IEEE. https://doi.org/10.1109/CSNT.2012.124

Briot, J.-P., Hadjeres, G., & Pachet, F.-D. (2019). Deep Learning Techniques for Music Generation—A Survey. https://doi.org/10.48550/arXiv.1709.01620

Chu, H., Kim, J., Kim, S., Lim, H., Lee, H., Jin, S., Lee, J., Kim, T., & Ko, S. (2022). An Empirical Study on How People Perceive AI-Generated Music. In *Proceedings of the 31st ACM International Conference on Information & Knowledge* (pp. 304-314). Association for Computing Machinery. https://doi.org/10.1145/3511808.3557235

Dua, M., Yadav, R., Mamgai, D., & Brodiya, S. (2020). An Improved RNN-LSTM Based Novel Approach for Sheet Music Generation. *Procedia Computer Science, 171,* 465-474. https://doi.org/10.1016/j.procs.2020.04.049

Hsu, J.-L., & Chang, S.-J. (2021). Generating Music Transition by Using a Transformer-Based Model. *Electronics, 10,* Article 2276. https://doi.org/10.3390/electronics10182276

Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Hawthorne, C., Dai, A., Hoffman, M. D., & Eck, D. (2019). *Music Transformer: Generating Music with Long-Term Structure*. https://magenta.tensorflow.org/music-transformer

Jagannathan, A., Chandrasekaran, B., Dutta, S., Patil, U. R., & Eirinaki, M. (2022). Original Music Generation Using Recurrent Neural Networks with Self-Attention. In *2022 IEEE International Conference On Artificial Intelligence Testing (AITest)* (pp. 56-63). IEEE. https://doi.org/10.1109/AITest55621.2022.00017

Ji, S., Luo, J., & Yang, X. (2020). A Comprehensive Survey on Deep Music Generation: Multi-Level Representations, Algorithms, Evaluations, and Future Directions. https://doi.org/10.48550/arXiv.2011.06801

Peracha, O. (2019). Improving Polyphonic Music Models with Feature-Rich Encoding. https://doi.org/10.48550/arXiv.1911.11775

Shaw, P., Uszkoreit, J., & Vaswani, A. (2018). Self-Attention with Relative Position Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Vo-*

lume 2 (Short Papers) (pp. 464-468). Association for Computational Linguistics. https://doi.org/10.18653/v1/N18-2074

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research, 15,* 1929-1958.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. https://doi.org/10.48550/arXiv.1706.03762