Original Music Generation using Recurrent Neural Networks with Self-Attention

Akash Jagannathan, Bharathi Chandrasekaran, Shubham Dutta, Uma Rameshgouda Patil, Magdalini Eirinaki
Computer Engineering Department
San José State University (SJSU)
San José, CA, USA
Email: {akash.jagannathan, bharathividhya.chandrasekaran, shubham.dutta, umarameshgouda.patil@sjsu.edu, magdalini.eirinaki}@sjsu.edu

***Abstract*—A recent trend in deep learning is using state-of-the-art models to generate human art forms. Using such "intelligent" models to generate novel musical compositions is a thriving area of research. The motivation is to use the capacity of deep learning architectures and training techniques to learn musical styles from arbitrary musical corpora automatically and then generate samples from the estimated distribution. We focus on two popular state-of-the-art models used in deep generative learning of music, namely recursive neural networks (RNN) and the self-attention mechanism. We provide a systematic evaluation of state-of-the-art models used in generative deep learning for music but also contribute novel architectures and compare them to the established baselines. The models are trained on piano compositions embedded in MIDI format from Google's Maestro dataset. A big challenge in such learning tasks is to evaluate the outcome of such learning tasks, since art is very subjective and hard to evaluate quantitatively. Therefore, in addition to the experimental evaluation, we also conduct a blind user study. We conclude that a double-stacked RNN model with a self-attention layer was observed to have the most optimal training time, and the pieces generated by a triple-stacked RNN model with self-attention layers were deemed the most subjectively appealing and authentic.**

***Index Terms*—Generative deep learning, music generation, MIDI, MAESTRO, piano-roll, RNN, Self-Attention**

I. Introduction

Music is an essential part of human culture that has existed from the earliest era of human civilization [1]. It has branched into various genres in different cultures, eras, and geographical regions. Appreciating music is a ubiquitous human characteristic. Traditionally, the musical note, a combination of beats, is generated by a human actor (or actors) using a musical instrument or human voice(s) [2]. However, in recent years, a significant amount of research has been conducted investigating automatic music generation using artificial intelligence [3].

Deep generative models have come into prominence in the past decade due to an increase in commercially available computational processing power and developments in machine learning algorithms. Typically, such models are neural networks with many hidden layers [4]. These models can be trained to capture salient features of a dataset by approximating high dimensional probability distributions. Then, they can be used to estimate the likelihood of a given sample and trained to generate new samples from the underlying distribution. Generative modeling has been applied to learn realistic distributions of text, visual and audio inputs and to generate new samples that are indistinguishable from the original samples. In this study we focus specifically on the use of deep learning models to generate novel musical compositions. Researchers have already proven the ability to develop artificial, music-generating pipelines using state-of-the-art deep learning models [3]. Techniques have been developed for generating piano sheet music, human voices, and timbres for various musical instruments [1]. Advancements in generative modeling have given rise to innovative ways of capturing the latent features of a musical dataset and generating new compositions from these features.

In this study, we provide a systematic evaluation of various state-of-the-art models used in generative deep learning for music. We also contribute a novel architecture and compare it to established baselines. Our efforts are focused on the following two models: recursive neural networks (RNNs) and self-attention. RNNs have been widely used to attempt to artificially compose music due to these models' ability to capture temporal dependencies in sequential data [5]. Self-attention is a recently developed technique that allows the "modeling of dependencies without regard to their distance in the input or output sequences" [6]. It allows for much greater parallelization than RNNs and thus is thought to train significantly faster [6]. Typically, self-attention is used either with a RNN or within a larger architecture known as a Transformer. In this study, we perform systematic evaluations on samples of music generated by three baselines: 1) a standalone self-attention model, 2) a RNN model, and 3) a RNN model that includes self-attention. We also evaluate two innovative models created by stacking RNN and self-attention layers on top of each other: a double-stacked model and a triple-stacked model. We train our models on a piano dataset [7] and evaluate artificial samples generated in a similar style.

The rest of the paper is organized as follows: we review related work in Section 2. Section 3 summarizes our models' architectures and information on other technical aspects of the project. Section 4 describes the experimental setup in this study and offers an analysis of our findings. Finally, we conclude the paper in Section 5 with a summary of findings, limitations of the present study, and plans for future work.

II. Related Work

Early research considered the LSTM algorithm to generate novel music because of its capability to recall past features and the structure of musical notes. Mangal et.al [2] examines a method for generating musical notes that employs recurrent neural networks (RNNs), specifically Long Short-Term Memory (LSTM) networks. Research shows that, in the generation of future notes, LSTMs have an advantage since they can recall past features and the structure of musical notes. Another direction of research involves the use of "chaotic inspiration" to assist LSTMs. This approach helps to avoid overfitting and to generate more attractive music, as illustrated in Coca et al. [8]. However, this architecture requires significant human involvement. Another LSTM-inspired architecture for generating music is revealed in Zhao et al. [9]. These researchers use bi-axial LSTM networks to generate polyphonic music, and include a concept known as LookBack into the architecture to improve the long-term structure of the compositions. Hewahi et.al [10] considered only Bach's musical style to train the neural network (NN) Model. LSTM is used in the core of the model and MIDI Files are taken as input to the model.

As we have discussed, an important aspect of music generation is the ability for the model to remember long-term structure. This aids in generating more meaningful, coherent content. Typically, RNN architecture such as LSTM has been used; however, its fixed memory size makes training notoriously difficult. An alternative approach known as self-attention has been considered, but is usually dismissed due to its reputation of scaling in quadratic time. The Magenta team at Google Brain were recently able to make the attention mechanism work in linear time, and released a model known as the MusicTransformer using this [11]. They were able to use this to generate minute-long sequences of meaningfully constructed musical compositions, which shattered previous length records of 15 to 30 seconds.

With the same intent to generate sustained music, Briot [3] provides a comprehensive review of contemporary deep learning techniques that automatically learn musical genres from arbitrary musical corpora. The authors highlight the usefulness of a popular new architecture known as a variational auto-encoder (VAE) to tackle the issue of creating sustained duration of meaningful content, and recommend using Generative Adversarial Networks (GANs) to create novel genres of music. The Google Brain project Magenta recently released a novel hierarchical VAE named MusicVAE that is successfully able to generate 30 seconds of novel, meaningful music [12]. Another Google project, DeepMind, recently released a VAE called VQ-VAE-2 [13]. They contributed the novel idea of stacking multiple VAEs that encode at different scales to generate better quality outputs. They proved this in the image generation domain, however, researchers have successfully applied this concept to music generation.

Boulanger-Lewandowski et al. [14] proposed a probabilistic model based on distribution estimators conditioned on a recurrent neural network (RNN) for detecting temporal dependencies in high-dimensional sequences. They utilize the piano-roll, a common representation for polyphonic music creation, which we use in our approach as well.

A sophisticated music generation tool combining all the aforementioned generative deep learning techniques is known as Jukebox [1]. While previous models are able to generate coherent music samples for 15 seconds to one minute in length, Jukebox's architecture exceeds this benchmark by generating semantically pleasing music samples that are multiple minutes long. The novel architecture consists of three separate deep learning algorithms: three stacked variational auto-encoders (VAE) encoding at different temporal resolutions, a simplified auto-regressive sequence model based on self-attention known as the Scalable Transformer, and dilated convolutions from the WaveNet CNN architecture [15]. A completely novel feature introduced by this model is the ability for it to work with lyrics and singing. Importantly, this model is able to work with raw audio waveforms. While for the purposes of our research we tried to replicate these industry-supported architectures and train our own models, we run into big limitations due to insufficient computing capacity. Unfortunately such models are impossible to use unless one has access to large-scale computing infrastructure and is beyond the scope of an academic project. We therefore focus our work on the architectures proposed in the research literature, namely self-attention models, and RNNs.

III. Model Architecture

A. *Self-Attention Model Baseline*

From motifs to phrases to portions like verse-chorus, a musical work frequently contains recurrent components at various levels. To build a cohesive work, a model must relate to pieces from the past, often from a long time ago, repeating, altering, and further developing them to create contrast and surprise. Self-attention [16] seems to be a good fit for this task. An auto regressive model may access any section of the previously created output at any phase of the production process thanks to self-attention over its own prior outputs. Recurrent neural networks on the other hand, must learn to proactively store pieces to be referenced in a fixed size state or memory, possibly complicating training.

Rather than concentrating on all of the data, the attention mechanism's goal is to emphasize significant concepts. We may highlight the significant tokens in a particular input sequence that are crucial for the model's predictions as well as execute feature selection using this method. This technique has been widely utilized in the vision community to focus on a specific part of an image with "high resolution" while perceiving the surrounding picture in "low resolution", and then modify the focal point over time.

1) Data Representation: We use a piano roll to represent music data. Because it can utilize rules to generate new music effectively, we merely generate notes and disregard other music performance metrics such as velocity.

2) Architecture: The Transformer decoder is a generative auto regressive model that relies on self-attention. Here we are using only one layer of self attention and then a dropout layer. The model in Fig. 1 has an encoder and a decoder. First the embedded input is fed to the Encoder which consists of multi head attention and feed-forward network. The output of the encoder is fed to a multi-head attention algorithm again following the feed forward network. We use the multiplicative attention type.

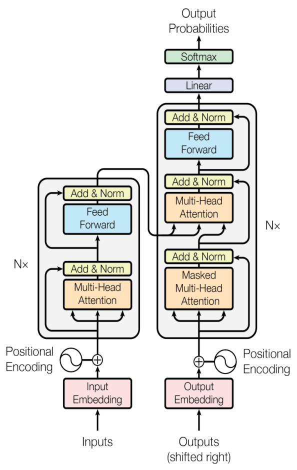

Fig. 1: Self Attention Model Architecture

Fig. 2 shows the model implemented for self attention baseline. Along with embedding and self attention, we also used a flatten function. The flatten function reduces input tensors from multiple dimensions to one dimension, which is required for us to perform further steps and train the model. Our source code reference for implementing self-attention can be found in [17].

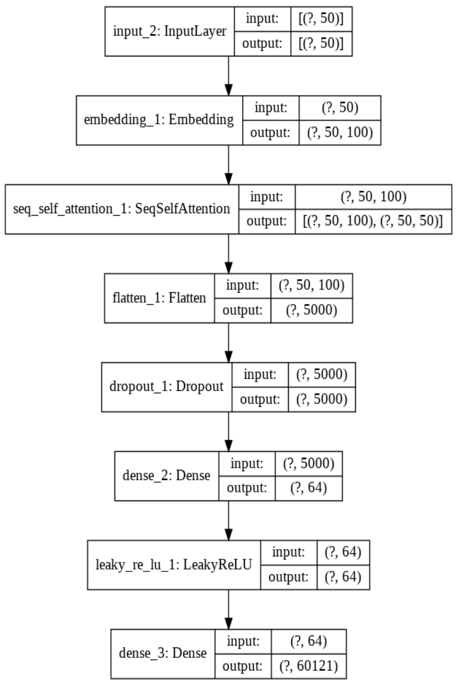
input_2: InputLayer | input: [(?, 50)] | output: [(?, 50)]
embedding_1: Embedding | input: (?, 50) | output: (?, 50, 100)
seq_self_attention_1: SeqSelfAttention | input: (?, 50, 100) | output: [(?, 50, 100), (?, 50, 50)]
flatten_1: Flatten | input: (?, 50, 100) | output: (?, 5000)
dropout_1: Dropout | input: (?, 5000) | output: (?, 5000)
dense_2: Dense | input: (?, 5000) | output: (?, 64)
leaky_re_lu_1: LeakyReLU | input: (?, 64) | output: (?, 64)
dense_3: Dense | input: (?, 64) | output: (?, 60121)

Fig. 2: Self Attention Model Baseline [6]

B. RNN Baseline

The difference of RNNs with simple NNs is that in this architecture there exists a recurrent, self-referring loop of the neurons in the hidden layer. This allows for capturing temporal dependencies in sequential data. In this type of model, inputs from previous time steps affect the way the neuron processes subsequent inputs. In such an architecture, temporal information only flows in the forward direction through time. However, researchers have shown qualitative improvements in music samples generated by bi-directional RNNs [18]. In such a model, temporal information flows in the negative direction as well, helping capture global dependencies in music.

For the RNN baseline (Fig. 3), we use a pipeline similar to the one described in the previous section. This helps us to make comparisons amongst the models. The input layer accepts our pre-processed MIDI files and feeds them to an embedding layer. The embedding layer converts our input data into a format acceptable by the bidirectional RNN.

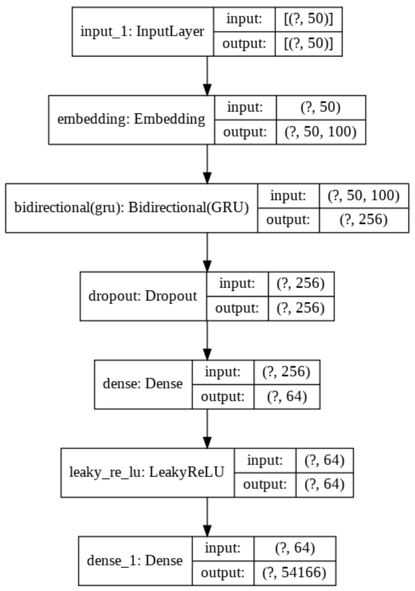
input_1: InputLayer | input: [(?, 50)] | output: [(?, 50)]
embedding: Embedding | input: (?, 50) | output: (?, 50, 100)
bidirectional(gru): Bidirectional(GRU) | input: (?, 50, 100) | output: (?, 256)
dropout: Dropout | input: (?, 256) | output: (?, 256)
dense: Dense | input: (?, 256) | output: (?, 64)
leaky_re_lu: LeakyReLU | input: (?, 64) | output: (?, 64)
dense_1: Dense | input: (?, 64) | output: (?, 54166)

Fig. 3: RNN Baseline Architecture

The reader will notice that the specific type of RNN we are using (as shown in Fig. 3) has a specialized implementation of neurons called GRUs (Gated Activation Unit). The architecture of a GRU is depicted in Fig. 4. A GRU uses gates within the neuron to better control information retention over longer time periods [19]. Standard RNN neurons without gates suffer from only capturing very short-term dependencies. The gates in a GRU solve this problem by allowing certain pieces of temporal information to be kept for longer, and other pieces of information to be released immediately.

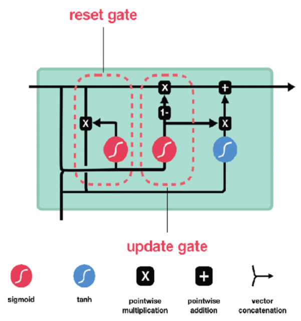

Fig. 4: GRU activation unit [19]

After the data passes through the bidirectional RNN consisting of GRU activation units, it is fed to a dropout layer to prevent over fitting. The last three parts of our architecture consist of a densely connected layer, a leaky RELU activation function, and another dense layer before our generated samples are outputted. These last three layers serve to recombine the latent features extracted by our RNN into new compositions. The loss function we use for training is the sparse categorical cross-entropy loss [17].

C. *RNN with Self-Attention*

As a third baseline model we have combined the significant parts of the previous two techniques. Specifically, we have created a deep learning pipeline containing a RNN (with GRU neurons) and a self-attention layer. As in the self-attention model, we include a flatten layer after the self-attention layer to transform the parameters into a format that is agreeable for subsequent layers of the model. Similar to the other two models, we include a dropout layer to avoid potential overfitting in the training phase. The full model pipeline is shown in Fig. 5.

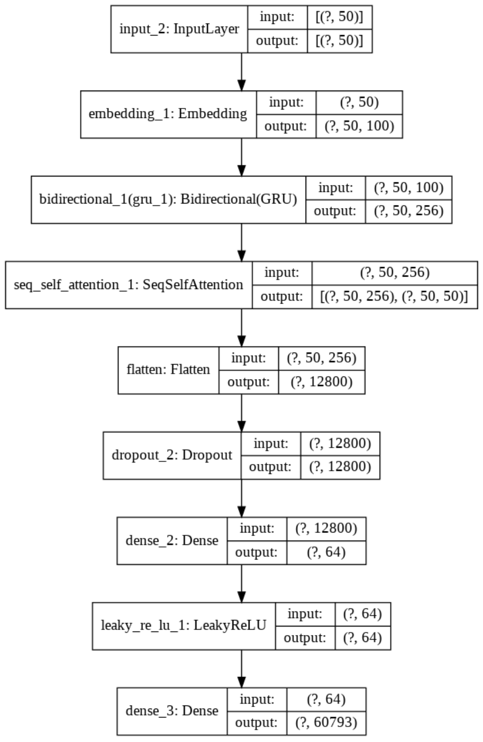
input_2: InputLayer | input: [(?, 50)] | output: [(?, 50)]
embedding_1: Embedding | input: (?, 50) | output: (?, 50, 100)
bidirectional_1(gru_1): Bidirectional(GRU) | input: (?, 50, 100) | output: (?, 50, 256)
seq_self_attention_1: SeqSelfAttention | input: (?, 50, 256) | output: [(?, 50, 256), (?, 50, 50)]
flatten: Flatten | input: (?, 50, 256) | output: (?, 12800)
dropout_2: Dropout | input: (?, 12800) | output: (?, 12800)
dense_2: Dense | input: (?, 12800) | output: (?, 64)
leaky_re_lu_1: LeakyReLU | input: (?, 64) | output: (?, 64)
dense_3: Dense | input: (?, 64) | output: (?, 60793)

Fig. 5: RNN with Self-Attention

D. *Beyond the baselines*

The previous three models described are used as baselines. Since we want to further explore whether combining the two state-of-the-art architectures can improve the music generation process, we propose two more architectures that build on the final baseline (RNN combined with self-attention)[1].

1) Double-Stacked RNN with Self-Attention: For the first proposed model architecture, we utilized a technique shown to be advantageous in [19]. The authors in that study compared various kinds of RNN architectures. They experimented with different neuron types within the RNN (i.e. the LSTM neuron architecture and the GRU neuron architecture). They also experimented with stacking RNNs on top of each other in the model pipeline. After performing subjective evaluations, they found that the double-stacked RNN using GRU neurons produced music that was rated to have the highest quality by human participants.

In this work, we extend the double-stacked RNN model, as shown in Fig. 6. We separate the layers of RNNs with a dropout layer. However, our model ultimately differs from [19] because after the first RNN stack, we include a self-attention layer. We did this to build upon our most robust baseline architecture and to separate our architecture from the one found in [19]. The last four layers of the model are kept consistent with the other models.

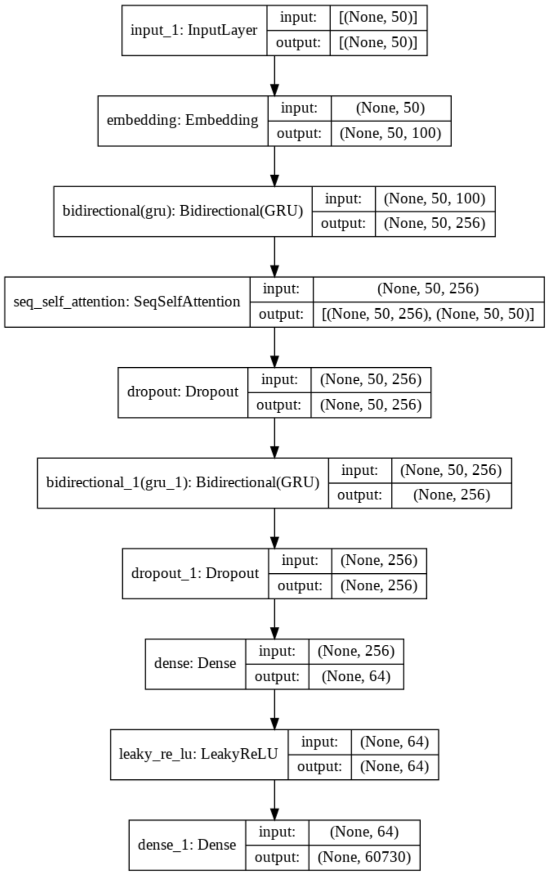
input_1: InputLayer | input: [(None, 50)] | output: [(None, 50)]
embedding: Embedding | input: (None, 50) | output: (None, 50, 100)
bidirectional(gru): Bidirectional(GRU) | input: (None, 50, 100) | output: (None, 50, 256)
seq_self_attention: SeqSelfAttention | input: (None, 50, 256) | output: [(None, 50, 256), (None, 50, 50)]
dropout: Dropout | input: (None, 50, 256) | output: (None, 50, 256)
bidirectional_1(gru_1): Bidirectional(GRU) | input: (None, 50, 256) | output: (None, 256)
dropout_1: Dropout | input: (None, 256) | output: (None, 256)
dense: Dense | input: (None, 256) | output: (None, 64)
leaky_re_lu: LeakyReLU | input: (None, 64) | output: (None, 64)
dense_1: Dense | input: (None, 64) | output: (None, 60730)

Fig. 6: Double-Stacked RNN with Self-Attention

2) Triple-Stacked RNN with Self-Attention: The second proposed model extension also draws inspiration from the results found in [19]. However, we take their findings a step further by introducing a triple-stacked RNN model architecture for producing novel musical rhythms. The model architecture is shown in Fig. 7. In an attempt to optimize our model's performance, we separate layers of RNNs by single dropout layers. This type of layout was found to be successful in [19] for a double-stacked RNN. Each of the first two layers of RNNs are followed immediately by a self-attention layer. After the stacking architecture, the final four layers are kept consistent with other models to facilitate meaningful comparisons. Our aim in building this architecture is to start to explore the

[1]Please note that we experimented with additional variations, but due to lack of space we present here the most successful ones.

extent to which stacking RNNs are useful in producing high-quality music.

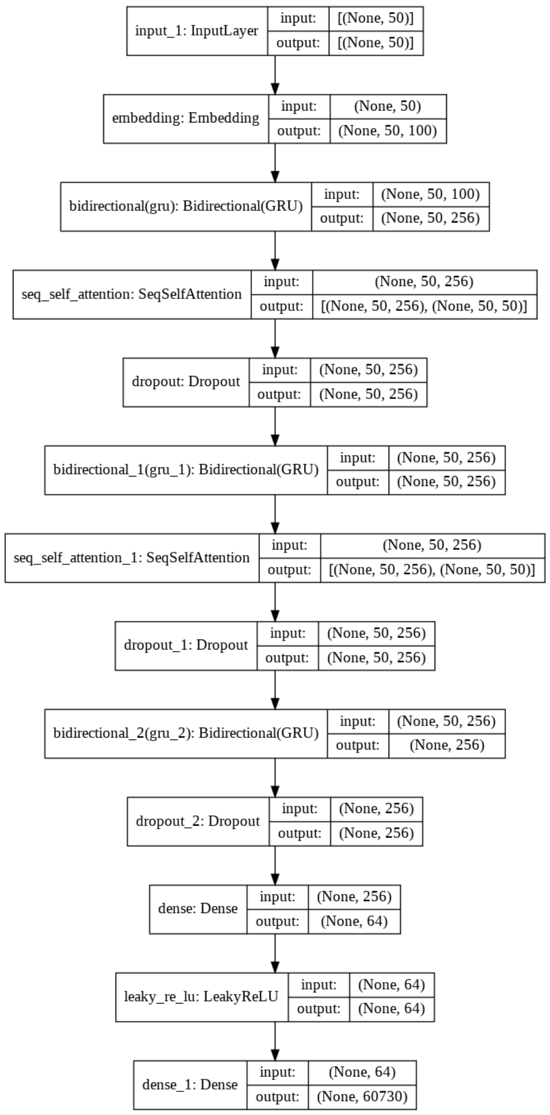
input_1: InputLayer | input: [(None, 50)] | output: [(None, 50)]
embedding: Embedding | input: (None, 50) | output: (None, 50, 100)
bidirectional(gru): Bidirectional(GRU) | input: (None, 50, 100) | output: (None, 50, 256)
seq_self_attention: SeqSelfAttention | input: (None, 50, 256) | output: [(None, 50, 256), (None, 50, 50)]
dropout: Dropout | input: (None, 50, 256) | output: (None, 50, 256)
bidirectional_1(gru_1): Bidirectional(GRU) | input: (None, 50, 256) | output: (None, 50, 256)
seq_self_attention_1: SeqSelfAttention | input: (None, 50, 256) | output: [(None, 50, 256), (None, 50, 50)]
dropout_1: Dropout | input: (None, 50, 256) | output: (None, 50, 256)
bidirectional_2(gru_2): Bidirectional(GRU) | input: (None, 50, 256) | output: (None, 256)
dropout_2: Dropout | input: (None, 256) | output: (None, 256)
dense: Dense | input: (None, 256) | output: (None, 64)
leaky_re_lu: LeakyReLU | input: (None, 64) | output: (None, 64)
dense_1: Dense | input: (None, 64) | output: (None, 60730)

Fig. 7: Triple-Stacked RNN with Self-Attention

IV. Evaluation

A. Dataset

In this study, we utilize Google's MAESTRO dataset [7]. The MAESTRO dataset contains over 200 hours of solo piano recordings. It includes 1282 real concert-quality acoustic grand piano performances of around 430 different pieces from the Yamaha piano e-competition. Each performance is available as a MIDI file and an audio recording aligned with a time resolution of about 3 ms. Metadata is available for each performance, including the composer and title of each. The dataset's size, fine alignment with ground truth, and the fact that it is real performance has made it an excellent source of training and evaluation data for artificially generating piano samples [20]. We sample 100 pieces randomly out of this dataset to train our auto-generative models.

B. Preprocessing

For musicians and producers, MIDI (Musical Instrument Digital Interface) is one of the most crucial tools. It's a communication protocol for computers, musical instruments, and other hardware. In simple words, a MIDI file consists of information about each instrument in a musical sample. It does not transmit the actual audio signal —it is information only. For instance, a sample can have the piano and guitar combo. Each musical instrument usually has specific notes to play. There are various libraries available to preprocess the MIDI files in python. *pretty_midi* is one of them. It can manipulate MIDI files as well as generate new ones.

In music theory, music consists of numerous tracks, each of which comprises multiple notes. Melody is a special track that is typically the essential component of music. This study employs a model to create a single melody track with the ability to play multiple notes simultaneously. In other words, it's a polyphonic music generation task. In MIDI, a note is a tuple consisting of $\langle velocity, pitch, start, end\rangle$. Start denotes the start of a note played in seconds. End marks the end of a note played in seconds. Multiple notes might overlap at the same time. Pitch is the MIDI number of the note played. Velocity is the force with which the note is played.

Piano-roll is an $N \times T \times C$ matrix, where N is the pitch count (N is 128 in MIDI, and the pitch ranges from 0 to 127), T is sequence length and time step count, and C is 2, which represents the length of each cell in the matrix [18]. In each cell, the first value indicates whether a note has been played (1 means played and 0 not played). On the other hand, the second value indicates whether a note is articulated (1 denotes that it is articulated, and 0 denotes otherwise). Piano-roll usually represents note sequences, but it ignores velocity (how hard the note was played). Velocity is a scalar quantity that spans from 0 to 127. This value is usually set to the highest volume in many MIDI files.As a result, we disregard velocity and assign it to a constant number.

In this paper, we need to extract all the notes of each instrument in each sample. Although MIDI files could have multiple instruments in their music, they contain only one instrument in our dataset: the *Piano*. We will extract the piano notes from the training samples. We extract the notes for the desired frame per second to make it easier. *pretty_midi* provides a useful function called get piano roll to retrieve the notes in a binary 2D array in the form of an array of (notes, time) dimensions. The notes' length is 128, and the time follows the duration of the music divided by FPS [17].

After we get the array of notes (piano-roll), we convert it into a dictionary. The dictionary begins from the time where the note is played. We then convert the dictionary into a sequence of notes which are fed as an input to the recurrent neural network. In addition, we set the next time-step to be the target for the input of the neural network.

C. Experimental Evaluation

Evaluating the results of generative models is a challenging task. Researchers are still working on defining quantitative metrics to evaluate the quality of artificially generated music samples. One model proposed in the literature is the Frechet

Inception Distance (FID) [21], an objective metric used for evaluating the quality of generated music samples.It measures the realism of the generated music samples compared to the original data set. Unfortunately, publicly available implementations of FID for music evaluation were unable to be accessed. Available implementations are geared towards image evaluation. Therefore, we could not include FID evaluations of our generated music in this study.

When training the models, we used TensorFlow's sparse categorical cross-entropy loss function. Recall that the input data format for our models was a dictionary of time and note pairs. Given a sequence of notes over time-steps, our models were trained to predict the most appropriate note (or beat of silence) at the next time-step. Notes in MIDI files are discretized and labeled by numbers ranging from 21 to 108 [17]. The sparse categorical cross-entropy loss is used to optimize our models' predictions to most closely match the next likely note. Utilizing this loss function, the loss over epochs for each our models can be seen in Fig. 8. The lowest loss over all epochs was displayed by the pure self-attention model (2), while the triple-stacked RNN with self-attention model had the highest loss over all epochs.

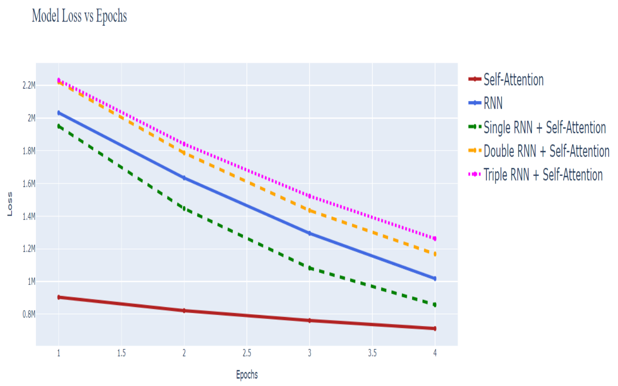

Fig. 8: Model Training: Sparse Categorical Entropy Loss over Epochs

We compared the training times of the 5 models. We used four epochs and a batch size of 16 samples. The training times for each of our five models is shown in Table I. The RNN baseline model took the longest to train (approximately 80 minutes), and the double-stacked RNN with self-attention had the fastest training time (approximately 18 minutes and 30 seconds). Surprisingly, the pure self-attention model and the triple-stacked RNN with self-attention had roughly equal training times at about 20 minutes.

Epochs = 4 (For all Models)	
Models	**Time Elapsed (minutes:seconds)**
RNN	79:39
Self-Attention	19:15
Single RNN + Self-Attention	26:42
Double RNN + Self-Attention	18:33
Triple RNN + Self-Attention	20:28

TABLE I: Model Training Times

We also conducted a user study to compare the piano samples generated by the pure RNN and the single-, double-, and triple-stacked RNN with self-attention models. In addition, we compared samples against original compositions used for training these models. We selected 30 participants for our study from our extended social circles[2]. The demographic of the participants spanned from 25 to 40 years of age. To design our user study we followed the paradigm of Walter et al. [21]. We divided the participants into three groups—A, B, and C. Each individual in a group was given eight music samples to rate on a scale of 1 to 5 (5 being the most positive). Each sample was 40 seconds long. All the participants in the same group got the same sample set but shuffled. The samples were rated based on three main aspects—*Appeal*, *Melody*, and *Realism*. In addition, the participants were asked to pick their favorite piece out of the sample set to give us a better understanding of how the models performed.

Participants in Group A got all the samples generated from the RNN models. We did not provide any information about whether the samples were artificially generated or composed by a human. On the other hand, participants in Group B got four samples generated by the RNN models and four original compositions. Like Group A, Group B didn't get any information about the samples. In addition, we asked Group A and B to label each piece if they were artificially generated (fake) or composed (real) based on their observation. Similarly, participants in Group C got four artificially generated samples and four original compositions. But this time, we appropriately labeled the samples (generated/composed) and asked the participants to rate how they genuinely felt about the compositions, knowing their true origin.

For Group A, the sample set contained 2 pieces generated by each model under consideration, as shown in Table II. *S.No* corresponds to the order in which the songs are presented in the survey and the *Song ID* corresponds to a unique song ID for each song. Song 3 (S.No 4), generated by the pure RNN model, was voted the most appealing by 60% participants. Song 11 (S.No 8), generated by triple-stacked RNN, was voted the most "real" song by 40% participants, and Song 5 (S.No 5), generated by single-stacked RNN, to be the most melodic by 70% participants. Song 11 (S.No 8) was labeled unanimously by 70% participants to be "composed by human", while in reality it was artificially generated. This song also happens to be the favorite of 50% of the participants.

Group A		
S.No	Song ID	Model
1	Song 7	Double stacked RNN + Self Attention
2	Song 8	Double stacked RNN + Self Attention
3	Song 4	Single stacked RNN + Self Attention
4	Song 3	Pure RNN
5	Song 5	Single stacked RNN + Self Attention
6	Song 10	Triple stacked RNN + Self Attention
7	Song 1	Pure RNN
8	Song 11	Triple stacked RNN + Self Attention

TABLE II: Group A sample set

For Group B, the sample set contained one piece generated by each model under consideration, as shown in Table III. Song 9 (S.No 1), generated by the double-stacked RNN model, and Song 11 (S.No 4), generated by the triple-stacked RNN model, were voted the most appealing to 60% participants

[2]The link to our survey is: https://forms.gle/igEDSYmLcgH9Cc2YA.

and the most melodic song by 50% participants. Song 11 was also judged to be the most "real" by 40% participants, and was labeled as "composed by human" by 60% participants, when in reality it was artificially generated. When the sample set contained the artificially generated samples and original compositions, we could observe that the participants were able to differentiate between them. Due to this reason, Group B's favorite song was one of the original compositions.

Group B		
S.No	Song ID	Model
1	Song 9	Double stacked RNN + Self Attention
2	Song 12	Original composition
3	Song 13	Original composition
4	Song 11	Triple stacked RNN + Self Attention
5	Song 14	Original composition
6	Song 6	Single stacked RNN + Self Attention
7	Song 15	Original composition
8	Song 2	Pure RNN

TABLE III: Group B sample set

Like Group B, the sample set of Group C also contains one piece generated by each model under consideration, as shown in Table IV. Song 10 (S.No 8), generated by the triple-stacked RNN model, was unanimously voted the most appealing by 50% participants, the most melodic song by 30% participants, and the most "real" song by 30% participants. 20% participants chose this song to be their favorite out of the lot despite knowing that these are artificially generated.

Group C		
S.No	Song ID	Model
1	Song 12	Original composition
2	Song 4	Single stacked RNN + Self Attention
3	Song 13	Original composition
4	Song 8	Double stacked RNN + Self Attention
5	Song 14	Original composition
6	Song 1	Pure RNN
7	Song 15	Original composition
8	Song 10	Triple stacked RNN + Self Attention

TABLE IV: Group C sample set

D. Results' Analysis

1) Training time: In terms of training time, the plain RNN model took the longest time to train, which was nearly 80 minutes. The model which used only the self-attention layer took significantly less time to train. Adding the self-attention layer to the RNN model also had significantly less training duration than the RNN model. Interestingly, training a double-stacked RNN with a self-attention layer had the least training time out of all the models. We found this to be a confusing and difficult to explain result. Intuitively, having more layers in the model should make the training process longer. We experimented with different order of the same layers, with no significant changes in the training times (and still all were faster than the plain RNN model). We were unable to come to a solid conclusion as to why our stacked models were some of the fastest to train. Future studies can look at the fine-grained mechanics of the deep learning algorithms within our models' pipelines to understand this finding.

2) Cross-entropy loss: We compared the models based on their aggregate loss after each epoch. We process our sampled songs note-by-note within each epoch of our models' training sessions. We refer to each note as a step. For each step, predictions are made using the sparse categorical cross-entropy loss function as to the most likely following note in the sequence. This prediction is compared to the actual next note, and the difference between the two equals the loss. This loss is aggregated over the entire epoch. We have trained all the models for 4 epochs with a batch size of 16.

We found the loss of all our models to be quite high. Our selection of the loss function might have contributed to this result. We used the sparse categorical cross-entropy measure to compare our models' predictions against real training values. However, another similar loss function is known as the categorical cross-entropy error. The difference between these two measures is that the sparse one is much more strict in judging correct predictions. It is effectively an all-or-nothing measure; if the model guesses the right note exactly, then there will be no loss. Otherwise, the model incurs a loss. In contrast, the non-sparse measure takes the likelihood of predictions into account. Therefore, it is a more forgiving measure. Utilizing this measure could result in decreased loss values for all our models.

Hybridized models with stacks of RNN and self-attention layers suffered the most sparse categorical cross-entropy loss during the initial epochs. However, the loss for these models decreases significantly during later epoch cycles when the model starts learning from the input data. Similarly, the RNN and RNN with self-attention models started with high loss values which significantly decreased over epochs. We observed that the model with only the self-attention layer suffered the least loss during training compared to the other 4 models. We believe this is because self-attention captures the global dependencies of the data. Thus, this model learned the data much faster but did not learn significantly more over successive iterations. Interestingly, the RNN with the self-attention model (non-stacked) suffers less loss in each epoch than the pure RNN model. This suggests that the self-attention layer of the former model leads to less loss. Perhaps, learning the global dependencies of songs helps in the task of sequential predictions of subsequent notes.

3) Additional architectures: A consideration in developing our proposed stacked models was to make them consistent with our third baseline (RNN with self-attention). However, the reader may pick up some key differences between the third baseline and our proposed models. First, the third baseline includes a flatten layer, while the two proposed models do not. Initially, we included flattened layers in our innovative models as well. However, we found that the addition of this layer lowered the subjective quality of the produced music. Unfortunately, we had to include the flattened layer in the third baseline to make the dimensions of the data agreeable for the last layers of the model. We experimented with a potential workaround to this. Specifically, we attempted to switch the order of the self-attention layer and the RNN layer in each of these models. This switch would exclude the need for a flattened layer to make the data format agreeable. However, switching this order of layers dramatically reduced music quality and increased training time. Therefore, we settled on the model architectures found in Section III.

4) User study: We have consolidated the results of Human Evaluation in Table V. In this table, the labels *A, M, and R* denote the parameters—*Appealing*, *Melodic*, and *Real*. We observe that the *triple-stacked RNN with self-attention* is voted to generate the most authentic and appealing pieces. This model could capture silences between notes very well compared to the other models, which mostly generated musical pieces with continuous notes.

Model	Group A	Group B	Group C
Pure RNN (Fig. 3)	A		
Single stacked RNN (Fig. 5)	M		
Double stacked RNN (Fig. 6)		A, M	
Triple stacked RNN (Fig. 7)	R	A, R	A, M, R

TABLE V: Overall Analysis of Human Evaluation

We further observe that the losses incurred by each model during training did not necessarily predict the subjective quality of the produced songs. The triple-stacked RNN with self-attention incurred the highest sparse categorical cross-entropy loss during training. However, this model also produced the highest quality outputs. The nature of music appreciation might explain this. Humans may not necessarily pay attention to the statistical "correctness" of the following note. Instead, they might appreciate meaningful notes. The triple-stacked RNN model may achieve this meaning due to two self-attention layers, which would more robustly capture global dependencies. Furthermore, the three layers of RNNs could capture the sequential data very meaningfully. Thus, the song outputted may be meaningful even though it is not algorithmically "correct". Finally, viewing the training times of our models in light of our subjective evaluations, we find that training time is not significantly correlated with subjectively-rated music quality.

V. Conclusions

In this paper, we have hybridized our model by using multiple stacks of RNNs and self-attention layers. The music samples of the piano rolls using our technique still hold the music's original structure. Most of our generated pieces from the stacked RNN with self-attention models were pleasing and melodic to the users. Samples generated by the tripled-stacked RNN with self-attention were found most authentic and appealing to the participants in the user study. Although the loss was high for our models, the addition of the self-attention layer dramatically decreased the training time and model loss. Getting feedback during the model development process could have helped us dynamically adjust our models in a meaningful direction. Future studies could follow an iterative process of deploying models. Specifically, changes can be made in the model pipeline or in tuning the hyperparameters of the existing models based on the feedback from subjective human evaluations.

In this study, we have stacked multiple RNNs with self-attention layers in our pipelines and found that the triple-stacked RNN model with two self-attention layers produced some of the highest subjectively-rated music out of all our generations. This extends the finding in [19] that double-stacking RNNs (with GRU neurons) produced better subjectively-rated music than a traditional, single-stacked model. Future studies could experiment with even more layers of stacks and additional loss functions.

References

[1] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," *arXiv preprint arXiv:2005.00341*, 2020.

[2] S. Mangal, R. Modak, and P. Joshi, "Lstm based music generation system," *arXiv preprint arXiv:1908.01080*, 2019.

[3] J.-P. Briot, "From artificial neural networks to deep learning for music generation: history, concepts and trends," *Neural Computing and Applications*, vol. 33, no. 1, pp. 39–65, 2021.

[4] L. Ruthotto and E. Haber, "An introduction to deep generative modeling," *CoRR*, vol. abs/2103.05180, 2021. [Online]. Available: https://arxiv.org/abs/2103.05180

[5] J.-P. Briot and F. Pachet, "Deep learning for music generation: challenges and directions," *Neural Computing and Applications*, vol. 32, no. 4, pp. 981–993, 2020.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[7] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, "Enabling factorized piano music modeling and generation with the MAESTRO dataset," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=r1lYRjC9F7

[8] A. E. Coca, D. C. Corrêa, and L. Zhao, "Computer-aided music composition with lstm neural network and chaotic inspiration," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2013, pp. 1–7.

[9] K. Zhao, S. Li, J. Cai, H. Wang, and J. Wang, "An emotional symbolic music generation system based on lstm networks," in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. IEEE, 2019, pp. 2039–2043.

[10] N. Hewahi, S. AlSaigal, and S. AlJanahi, "Generation of music pieces using machine learning: long short-term memory neural networks approach," *Arab Journal of Basic and Applied Sciences*, vol. 26, no. 1, pp. 397–413, 2019.

[11] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer," *arXiv preprint arXiv:1809.04281*, 2018.

[12] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A hierarchical latent vector model for learning long-term structure in music," in *International conference on machine learning*. PMLR, 2018, pp. 4364–4373.

[13] A. Razavi, A. van den Oord, and O. Vinyals, "Generating diverse high-fidelity images with vq-vae-2," in *Advances in neural information processing systems*, 2019, pp. 14 866–14 876.

[14] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," *arXiv preprint arXiv:1206.6392*, 2012.

[15] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[16] A. P. Parikh, O. Täckström, D. Das, and J. Uszkoreit, "A decomposable attention model for natural language inference," *arXiv preprint arXiv:1606.01933*, 2016.

[17] H. A. Wibowo, "Generate piano instrumental music by using deep learning," Mar 2019. [Online]. Available: https://towardsdatascience.com/generate-piano-instrumental-music-by-using-deep-learning-80ac35cdbd2e

[18] T. Jiang, Q. Xiao, and X. Yin, "Music generation using bidirectional recurrent network," in *2019 IEEE 2nd International Conference on Electronics Technology (ICET)*. IEEE, 2019, pp. 564–569.

[19] A. A. S. Gunawan, A. P. Iman, and D. Suhartono, "Automatic music generator using recurrent neural network," *International Journal of Computational Intelligence Systems*, vol. 13, no. 1, pp. 645–654, 2020.

[20] F. Foscarin, A. Mcleod, P. Rigaux, F. Jacquemard, and M. Sakai, "Asap: a dataset of aligned scores and performances for piano transcription," in *International Society for Music Information Retrieval Conference*, no. CONF, 2020, pp. 534–541.

[21] S. Walter, G. Mougeot, Y. Sun, L. Jiang, K.-M. Chao, and H. Cai, "Midipgan: A progressive gan approach to midi generation," in *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2021, pp. 1166–1171.