Music Generation using GAN

Chinmai Kaidabettu Srinivas (ckaidab)
Department of Computer Science
North Carolina State Univerity
Raleigh, United States
ckaidab@ncsu.edu

Nikita Joshi (njoshi2)
Department of Computer Science
North Carolina State University
Raleigh, United States
njoshi2@ncsu.edu

Pooja Patel (pkpatel5)
Department of Computer Science
North Carolina State University
Raleigh, United States
pkpatel5@ncsu.edu

I. Introduction

Today, neural networks are prevalent in solving problems pertaining to text, image and video classification. As opposed to these, music is a relatively unexplored field offering a newer set of learning opportunities and posing more complex challenges. Generating realistic and aesthetic music is one of the major challenging part of this task. Music generation requires extensive exploration of the patterns in the music. Music has some hierarchical structure which we should explore and understand before we generate our own music. Also generally music is composed of multiple instruments and tracks which is very hard to understand. However, it can have exciting applications for consumers ranging from commercial media companies to music composers. Employing neural networks to synthesize new data to imitate human talents is yet an emerging application. In this project, we aim to generate human-comparable music using a more advanced GAN architecture. We used a subset of the MAESTRO dataset which is composed of about 200 hours of piano music. For each file, the metadata available in this dataset includes information such as Composer, Year, MIDI filename and Duration. This dataset is available for download from *here.* Some of the challenges for the task of Music Generation are:

- Each piece of music is unique in its own way. So our model should be good enough to identify the nuances in the music.
- Evaluating the quality of the music that is generated by our model.
- Musical data would require a better understanding of musical technicalities to pre-process the data properly.

II. Methodology

Below is the high-level description of the approach that our team followed:

- Pre-process the midi files in order to extract sequence of notes.
- Creation and evaluation of baseline LSTM model.
- Creation and evaluation of GAN model.

Our approach to the problem involves following steps:

A. Pre-processing and Analysis:

We pre-processed the music files which is in the MIDI format to extract all the notes and chords present in the midi file. We used music21 library in Python to accomplish this task. We identified all the unique notes and chords present in our training set and assigned an unique integer value to each of those unique notes and chords. Based on this mapping, we then converted are musical notes to a sequence which can be input to our neural network models. Figure 1 shows a pictorial representation of this process.

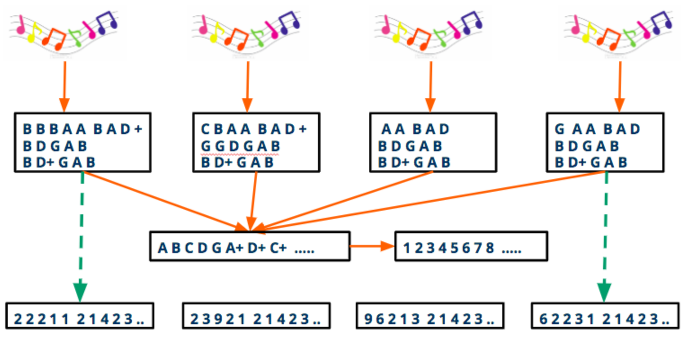

Fig. 1. Data Pre-processing

B. Creation and Evaluation of baseline model:

We created a baseline model using LSTM as mentioned in this article. LSTM is a recurrent neural network architecture that captures information sequentially over time rather than space. LSTMs are well-suited for classifying, processing and making predictions based on the time series data and can handle noise, distributed representations and continuous values. Moreover, LSTMs can remember and reuse inputs over a long period of time which is a very good property in terms of our problem statement.
The first layer in the network was the main LSTM layer to which we pass our input sequence and it was the main layer as part of training of network on how to predict notes in a certain sequence. The second layer was a dropout layer which was basically used to down sample our output and hence prevent over fitting. This was followed by a series of LSTM and dropout layers. The output of the last dropout layer was then passed on the fully connected layer.The activation function we used in our network was ReLU.
Figure 2 describes the network architecture structure for our LSTM model

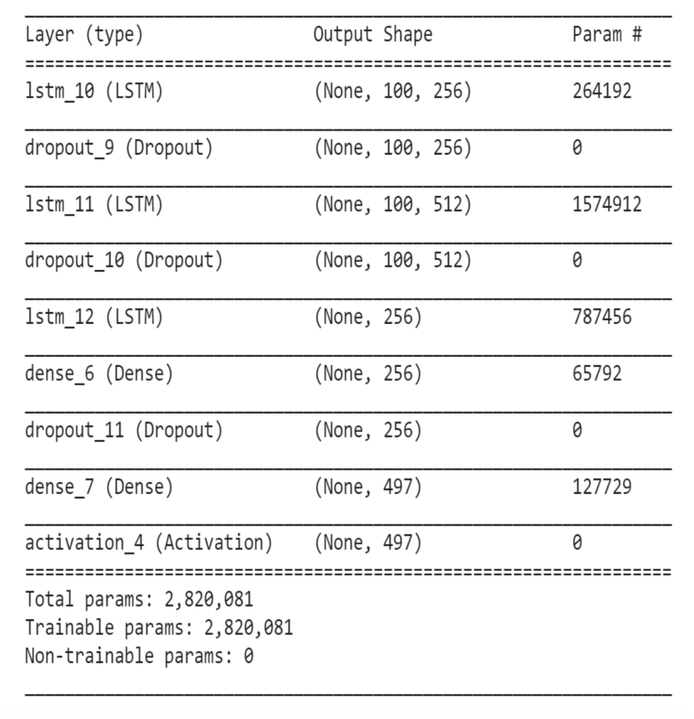
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_10 (LSTM)               (None, 100, 256)          264192
_________________________________________________________________
dropout_9 (Dropout)          (None, 100, 256)          0
_________________________________________________________________
lstm_11 (LSTM)               (None, 100, 512)          1574912
_________________________________________________________________
dropout_10 (Dropout)         (None, 100, 512)          0
_________________________________________________________________
lstm_12 (LSTM)               (None, 256)               787456
_________________________________________________________________
dense_6 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_11 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 497)               127729
_________________________________________________________________
activation_4 (Activation)    (None, 497)               0
=================================================================
Total params: 2,820,081
Trainable params: 2,820,081
Non-trainable params: 0
_________________________________________________________________
```

Fig. 2. LSTM architecture

C. *Creation and Evaluation of GAN:*

After building our baseline LSTM model, we trained an adversarial network to try and achieve better results at the same task. The Generative Adversarial Network(GAN) consisted of a generator model whose task was to generate a music from a randomly generated noise. Another component of our GAN was a discriminator which was given two files as an input simultaneously, an actual real music file which was present in the dataset and a fake music file generated by the generator. The discriminator's task was to differentiate between the real music file and the fake music file. The setup was aimed at creating a set of adversarial models that compete against each other to achieve better results until the generator gets so good that it is able to fool the discriminator. Figure 3 shows the general architecture of GAN. The process of training GAN can be formalized as a two-player minimax game between the generator and the discriminator.

Generator network:
The generator network was a simple multi-layered perceptron that received random inputs equal to the size of the latent dimension. As mentioned earlier, its task was to generate music sequence from the given random noise. Figure 4 describes the architecture of our generator model.

Discriminator network:
The first two layers in the discriminator model were LSTMs, one of which was a bidirectional LSTM so that the model can take context in both directions into account to make it's decision. Figure 5 describes the architecture of our discriminator model.

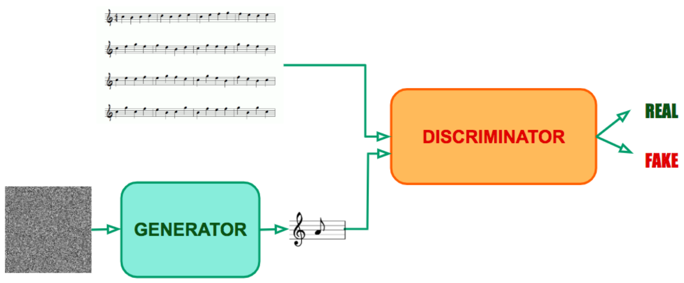

Fig. 3. GAN architecture

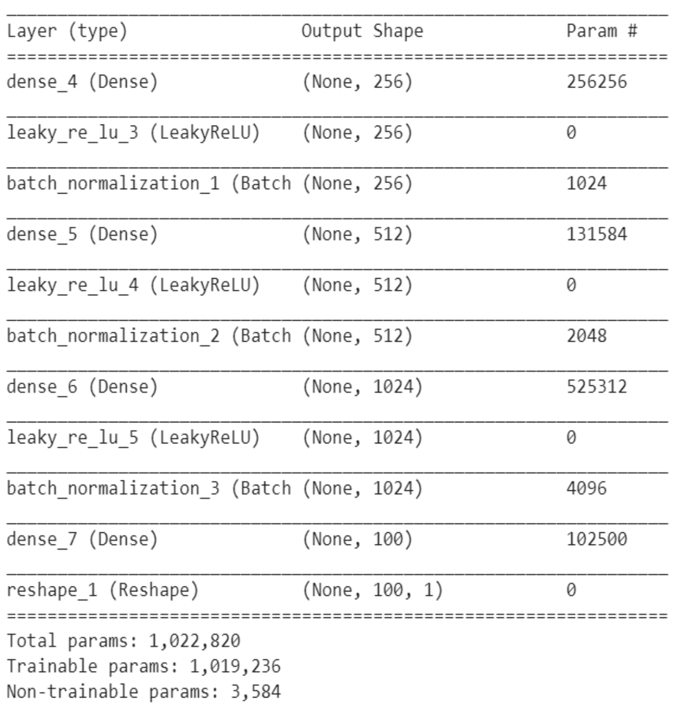
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 256)               256256
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
dense_5 (Dense)              (None, 512)               131584
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
batch_normalization_2 (Batch (None, 512)               2048
_________________________________________________________________
dense_6 (Dense)              (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
batch_normalization_3 (Batch (None, 1024)              4096
_________________________________________________________________
dense_7 (Dense)              (None, 100)               102500
_________________________________________________________________
reshape_1 (Reshape)          (None, 100, 1)            0
=================================================================
Total params: 1,022,820
Trainable params: 1,019,236
Non-trainable params: 3,584
```

Fig. 4. Generator architecture of GAN

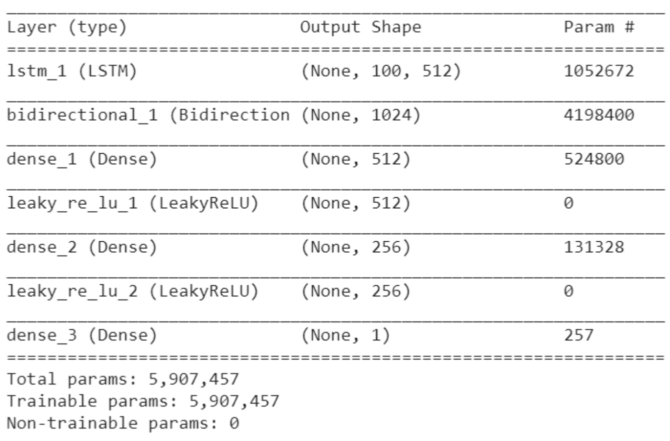
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 100, 512)          1052672
_________________________________________________________________
bidirectional_1 (Bidirection (None, 1024)              4198400
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 257
=================================================================
Total params: 5,907,457
Trainable params: 5,907,457
Non-trainable params: 0
```

Fig. 5. Discriminator architecture of GAN

III. Model Training and Hyper-parameter selection

A. Model Training

We first trained the LSTM model and generated the music sequence from a given randomly generated input. We then calculated the percentage of excessively repeated notes and also the range of music notes generated which are our primary evaluation metrics (explained in detail in the section IV). Figure 6 shows the loss per epoch for the Baseline LSTM model.

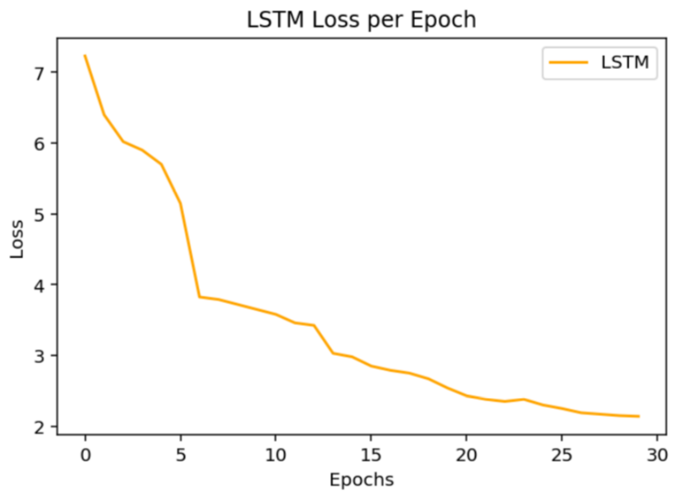

Fig. 6. Loss per epoch for our baseline LSTM model

Once we had the results for our baseline model, we then trained our GAN model. The GAN architecture that we built was trained as mentioned below:

- We generated a fake music sequence from the generator by passing a randomly generated sequence to it.
- The generated fake music sequence and the actual pre-processed music sequence from the dataset was passed as input to the discriminator.
- Both generator and discriminator model's parameters will be updated where the generator tries to get better in fooling the discriminator, and the discriminator tries to get better in differentiating between the real and fake music sequences.

Figure 7 shows the loss per epoch for the GAN model.

B. Hyper-Parameter Tuning

How well a model performs depends not only on its architecture but largely on the fine details of the architecture – the hyper parameters. To find the best possible model, we have fine-tuned the hyper parameters of the GAN model (both generator and discriminator) by testing them out on a range of values and picking the optimal value in order to achieve the best model performance in terms of percentage of notes that are excessively repeated and range of notes generated (explained in detail in section IV). The various hyper parameters that we have selected for fine-tuning are as follows:

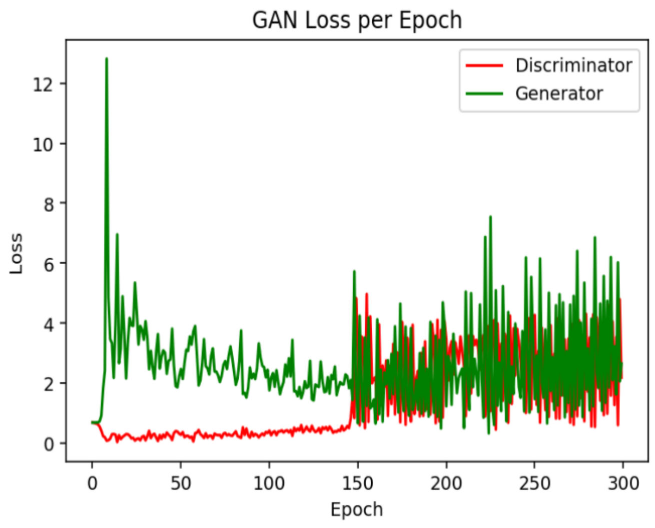

Fig. 7. Loss per epoch for the GAN model

- Activation Function: [relu, leakyRelu]
- Learning Rate: [0.01, 0.001, 0.1]
- Optimizer: [Adam, RMSProp, Adadelta]
- Batch Size: [64, 128, 256]

1) Activation Function: We initially fixed a learning rate as 0.01, batch size as 64 and optimizer as RMSProp (as these were the hyper parameters in our baseline model) and trained our model for different activation functions as mentioned above. Table I shows the comparison of the percentage of excessively repeated notes and range of notes generated for different activation functions. We got the best values when we chose LeakyRelu as the activation function.

TABLE I
PERCENTAGE OF EXCESSIVELY REPEATED NOTES AND RANGE FOR ACTIVATION FUNCTION

Activation Function	Notes excessively repeated(%)	Range
Relu	58	56
LeakyRelu	56	56

2) Learning Rate: Next, we tuned the learning rate by fixing activation function as the LeakyRelu and Adadelta as the optimizer. Table II shows the percentage of excessively repeated notes and range of music notes generated for different learning rates. We obtained best values for learning rate of 0.01.

TABLE II
PERCENTAGE OF EXCESSIVELY REPEATED NOTES AND RANGE FOR DIFFERENT LEARNING RATE

Learning Rate	Notes excessively repeated (%)	Range
0.1	57	51
0.01	56	57
0.001	60	53

3) Optimizer: We then tuned the optimizer by fixing the learning rate as 0.01 and activation function as LeakyRelu.

Table III shows the percentage of excessively repeated notes and range of music notes generated for different optimizer. We obtained best values when we chose Adam as the optimizer.

TABLE III
PERCENTAGE OF EXCESSIVELY REPEATED NOTES AND RANGE FOR DIFFERENT OPTIMIZERS

Optimizer	**Notes excessively repeated(%)**	**Range**
Adam	58	55
Adadelta	59	51
RMSProp	61	49

4) Batch Size: Next, we tuned the batch size by fixing the best hyper parameters obtained so far, i.e. learning rate as 0.01, activation function as LeakyRelu and optimizer as Adam. Table IV shows the percentage of excessively repeated notes and range of music notes generated for different optimizer. We obtained best values for a batch size of 128.

TABLE IV
PERCENTAGE OF EXCESSIVELY REPEATED NOTES AND RANGE FOR DIFFERENT BATCH SIZE

Batch Size	**Notes excessively repeated(%)**	**Range**
64	57	59
128	56	57
256	60	62

We trained the model again by choosing all the best hyper parameters, i.e. learning rate as 0.01, activation function as LeakyRelu, optimizer as Adam, a batch size of 128 and obtained a loss curve as show in Figure 7.

IV. EVALUATION

Evaluation of the generated music is a challenging task as music generation is an unsupervised learning process. The best way to evaluate the model would be human evaluation. In this process, the output so generated from the model gets reviewed by humans. Although this is one of the best evaluation metrics, it might be impossible to get all the samples generated by our model to be evaluated by humans. Hence, we used some quantitative measures derived from music theory rules to judge the aesthetic appeal of the model generated music. Below-mentioned are some of the measures for performing a quantitative analysis of our model.

- Notes excessively repeated: This is the measure of the fraction of notes that are most redundant in a piece of music. This is a good metric to check for the amount of variability of the generated music. A high amount of repetition would mean that the model is not generating new sequences and has no variability. The generative model might just generate same sequence of high probability music sequences without learning anything which means that the generated music has no variability.
- Range: This metric checks for the distance between a tune's highest and lowest note in terms of tone steps. As per the general music aesthetics, music generated with a tone-span around 36 is considered as good music.

The table V below summarizes the results that we got for these metrics for both baseline LSTM and GAN model:

TABLE V
COMPARISON OF BASELINE LSTM MODEL AND GAN USING EVALUATION METRICS

	Notes Excessively Repeated (%)	**Range**
Baseline LSTM	82	32
GAN	55	56

From Table V, we can see that the baseline model had more percentage of repeating notes as compared to our GAN model. We generated multiple music files from both the baseline and GAN model. Next, we calculated the average percentage of notes that were excessively repeated. On an average, we observed that 82 percent of notes were being repeated in the baseline model while only 55 percent of the notes were being repeated in the GAN model. The range metric gave slightly different results. The baseline LSTM model had a range of 32 whereas the GAN model had a range of 56. Our intuition for this is that the baseline model had softmax activation function in its final layer which generates highest probable note for a given sequence. Also, the LSTM model is generating repetitive music because of which the difference between the highest and lowest note comes down better as compared to GAN. In reality GAN was able to generate better note with variations compared to that of LSTM.

Along with the qualitative metrics, we also performed a survey to analyze how good the music generated using our proposed GAN model compared to that of the baseline model. Several Humans with basic knowledge about the piano music were chosen for this survey. According to the survey results 99 percent of the people mentioned that the music generated using GAN was better as compared to the baseline LSTM model.

The main idea behind using the above mentioned statistical measures along with human evaluation was to make sure that the music generated by the model adheres to the music on which the model was trained on. We believe that our presented results clearly indicate that the proposed model GAN outperformed the baseline LSTM model. Also, our proposed GAN model was able to generate music which had unique tones and was good enough to identify the nuances in the music.

REFERENCES

[1] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel and Douglas Eck "Enabling Factorized Piano Music Modelling And Generation With The Maestro Dataset" ICLR , Jan 2019.
[2] Natasha Jaques, Shixiang Gu, Richard E. Turner, Douglas Eck "Generating Music by Fine-Tuning Recurrent Neural Networks with Reinforcement Learning"
[3] Allen Huang, Raymond Wu "Deep Learning for Music" Stanford University