Music Generation System using Deep Learning

Dhana Lakshmi Kankanala

Harshitha Paladugu Chengalraya

Abstract:

In the field of artificial intelligence and machine learning, music generation is one of the most fascinating areas of research. Long Short-term memory (LSTM) and Gated Recurrent Unit (GRU) which are a type of Recurrent Neural Network (RNN) models are used to generate the music using the MIDI file dataset. The objective is to produce expressive and well-structured, simple piano compositions that express the distinctive characteristics of the instrument. The project analyzes MIDI data preprocessing, LSTM, GRU architecture designs for music synthesis, and assessment of the produced music using both objective metrics and empathetic assessments. Future developments in AI-assisted piano music creation are made possible by addressing the difficulties of retaining diversity, combating overfitting, and improving musical quality.

Introduction:

The integration of technology and creativity has recently contributed to significant advances in the fields of music and artificial intelligence. This research utilizes Long Short-Term Memory (LSTM) networks and Gated Recurrent Unit (GRU) to explore the fascinating field of piano music generation for individual artists.

Notes, chords, and Octave are some fundamental notations for the piano instrument used in this project.On a piano, a note is the sound made by pressing just one key. The building elements of music are notes, and each note has a unique sound.The group of notes played collectively is referred to as a "chord." Chords are like a set of notes that sound well when performed together; they provide music depth and harmony.Two notes that share the same pitch are separated by an octave.

Challenges:

Musical data is often represented as sequences of notes, chords, and other musical elements, resulting in input spaces with many dimensions. Managing and processing such data requires

careful design to avoid issues such as vanishing and exploding gradients, which can create difficulties for learning. There is a possibility that models will memorize existing compositions rather than create new music. It is a delicate task to balance the exploration of new musical ideas while avoiding overfitting the training data. MIDI data is frequently sparse, having several rests or gaps between notes. Due to the sparsity of the input, this can provide problems when training LSTM or GRU models, as they may have difficulties learning long-term dependencies.

Dataset:

Due to their unique characteristics and adaptability, MIDI (Musical Instrument Digital Interface) files are often used in numerous kinds of music-related applications. Multiple instrument tracks can be represented in one MIDI file. The tracks' separation enables separate modification of the notes and attributes of each instrument. Models can now produce music with authenticity and style by using MIDI files, which encode essential musical elements like notes, chords, and tempo. Deep learning models for creating music are trained and evaluated using this data.

A collection of MIDI files that have been selected and arranged according to the musical genres and compositions of specific artists is referred to as an artist-specific MIDI dataset.

The digital format MIDI is an effective choice for storing musical data since it can express musical notes, durations, velocities, and other properties. Each MIDI file contained in an artist-specific MIDI dataset MIDI dataset is associated to a certain composer. These MIDI files, which represent the artist's creative output, can include portions from different genres, time periods, or moods. This type of dataset is useful for training machine learning models, such LSTM or GRU networks, to produce music that accurately mimics the distinctive styles and characteristics of many artists.

The MIDI dataset for this project consists of 25 artist folders (Figure 1(a)), each of which has an average of 15 midi files with durations ranging from one minute thirty seconds to two minutes.

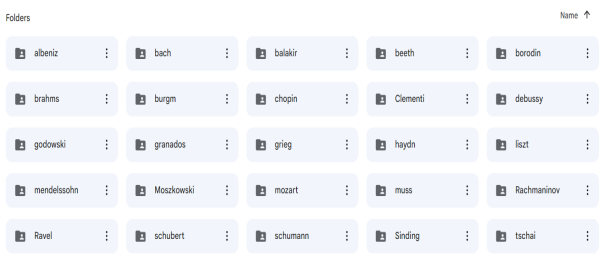


Figure 1(a)

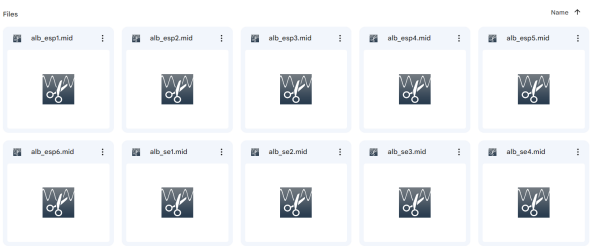


Figure 1(b)

Related Work:

Google's Magenta project: A wide range of tools and models for AI-generated music have been provided by Magenta, exhibiting the potential of LSTM networks to create complex musical compositions. Magenta analyzes patterns in existing music and generates new compositions that follow those patterns using deep learning techniques, such as LSTM networks. Magenta's findings are incredible, but it's still difficult to create original, deeply moving music that compares to the best work of human artists. The created music may also tend to lack a unique thematic framework or emotional richness.

MuseNet by OpenAI: MuseNet has shown the versatility and flexibility of AI-generated compositions by displaying the possibilities of LSTM networks to generate music across a wide variety of genres and styles. To learn from a wide range of musical input and produce coherent and emotive compositions, MuseNet uses large-scale neural networks, including LSTM. Despite its adaptability, MuseNet's produced music can occasionally come out as overly repetitive or predictable. Additionally, it may have trouble with some difficult musical structures that call for in-depth creative interpretation.

Reading and Parsing the Data:

This project involves reading and parsing MIDI files containing musical compositions in order to retrieve relevant musical data. To express musical notes, durations, velocities, and other elements, MIDI files use a digital format. In this project, we parse MIDI files using the music21 library. The main objective is to extract the notes, octave, and chords from the MIDI files' piano-specific data. Also implement the rest during the encoding of the midi file data into numerical input for the model.

Investigating and Processing the Dataset: After MIDI file parsing, the project investigates and preprocesses the dataset in order to use it for model training. This can be done in few steps:

Unique Notes and frequency: The code determines the frequency at which each of the dataset's unique notes occur. This aids in analyzing note distribution and identifying patterns in data.

```
Unique Notes: 723
30 : 249
50 : 225
70 : 209
90 : 202
```

To improve the training efficiency and focus on frequently occurring notes, a threshold frequency is chosen. Notes with frequencies below this threshold are filtered out, resulting in a refined dataset. This step also helps in addressing the challenge of data sparsity.

Creating Input and Output Sequences: Once the dataset is refined, the next step is to create input and output sequences for training the LSTM model. This involves the following steps:

Sliding Window Approach: A sliding window approach is used to create sequences of notes. A fixed number of notes (timesteps) are taken as input, and the following note is taken as the output. This process generates pairs of input sequences and corresponding target outputs.

Mapping to Indices: The notes are mapped to the correct index values to make the data acceptable for model training. To convert notes to indexes and vice versa, dictionaries are made. The model can now operate with numerical data because of this.

Model

LSTM Model:

A Long Short-Term Memory (LSTM) model is a type of recurrent neural network (RNN) architecture that is designed to capture and learn long-range dependencies and patterns in sequential data. The primary reason for using an LSTM model in our music generation project, particularly when working with MIDI datasets for piano music generation, is its ability to capture temporal relationships and complex patterns in sequences.

LSTMs are specifically designed to address the vanishing gradient problem that plagues standard RNNs. They have an internal memory cell that can store information over long sequences, allowing the network to capture relationships between distant notes in the music, which is crucial for creating coherent and melodically rich compositions.

Layer (type)	Output Shape	Param #
lstm_5 (LSTM)	(None, 50, 256)	264192
dropout_5 (Dropout)	(None, 50, 256)	0
lstm_6 (LSTM)	(None, 50, 512)	1574912
dropout_6 (Dropout)	(None, 50, 512)	0
lstm_7 (LSTM)	(None, 256)	787456
dense_4 (Dense)	(None, 256)	65792
dropout_7 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 65)	16705

Total params: 2,709,057
Trainable params: 2,709,057
Non-trainable params: 0

Figure 2: LSTM model architecture

This LSTM model is composed of three layers. For LSTM layer 1,It has 256 LSTM units, taking sequences of length 50 and processing them. This layer has 264,192 trainable parameters. The batch normalization normalizes the output of LSTM Layer 1, improving convergence during training. Applies dropout regularization to the output of every Batch Normalization, preventing overfitting.

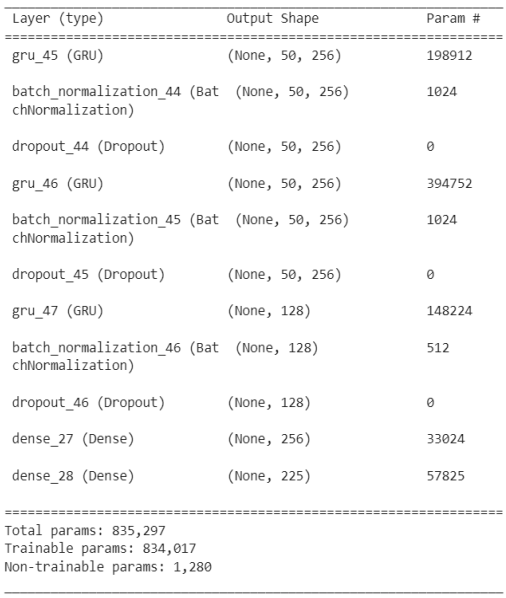
Another LSTM layer with 256 units, also processing sequences of length 50. It has 525,312 trainable parameters.A final LSTM layer with 128 units, processing the output sequence from the previous layers. It has 197,120 trainable parameters. A fully connected dense layer with 256 units and a ReLU activation function, contributing 33,024 trainable parameters. The final dense layer with 65 units (matching the number of unique notes or classes),using a softmax activation function to generate probabilities. It has 16,705 trainable parameters.

The model has a total of 1,038,913 parameters, out of which 1,037,633 are trainable and can be learned during training. The architecture combines LSTM layers for sequence processing, batch normalization for stabilization, dropout for regularization, and dense layers for output prediction. This model is designed for generating sequences of musical notes, with the last dense layer producing a probability distribution over the possible next notes in the sequence.

GRU Model:

A recurrent neural network (RNN) architecture called a GRU (Gated Recurrent Unit) would be ideal for sequential data tasks like generating music from MIDI files. It offers an acceptable balance between capturing long-term dependencies and computational efficiency, addressing some of the shortcomings of ordinary RNNs and LSTM (Long Short-Term Memory) networks. GRUs are a good fit for this project because they can handle issues like memory optimization, vanishing gradients, and overfitting while capturing the sequential relationships and patterns in musical compositions.

The model consists of several layers, including GRU layers, Batch Normalization layers, Dropout layers, and Dense (fully connected) layers.



Layer (type)	Output Shape	Param #
gru_45 (GRU)	(None, 50, 256)	198912
batch_normalization_44 (BatchNormalization)	(None, 50, 256)	1024
dropout_44 (Dropout)	(None, 50, 256)	0
gru_46 (GRU)	(None, 50, 256)	394752
batch_normalization_45 (BatchNormalization)	(None, 50, 256)	1024
dropout_45 (Dropout)	(None, 50, 256)	0
gru_47 (GRU)	(None, 128)	148224
batch_normalization_46 (BatchNormalization)	(None, 128)	512
dropout_46 (Dropout)	(None, 128)	0
dense_27 (Dense)	(None, 256)	33024
dense_28 (Dense)	(None, 225)	57825

Total params: 835,297
Trainable params: 834,017
Non-trainable params: 1,280

Figure 3: GRU model architecture

The model begins with a 256-unit GRU layer. A sequence of the same length, but with 256 dimensions, is produced by this layer after processing sequences of length 50 (timesteps). A second GRU layer, again with 256 units, is added thereafter to continue processing the sequence. After every GRU layer, batch normalization is used to stabilize and expedite training by ensuring that each layer's output is normal. To avoid overfitting, dropouts are used after batch normalization. During training, a certain number of neurons are randomly dropped, which encourages the model to acquire more robust properties.

Following that is a third GRU layer with 128 units. This layer performs additional processing on the sequence and produces a 128-dimensional sequence. The output is then processed via a dense layer with 256 units and a ReLU activation function by the model. The model's output is generated by the last dense layer, which contains 225 units (equivalent to the number of distinct notes and chords in the dataset) and implements the softmax activation function.

Modifications needed for data:

Depending on the complexity of the music pieces and the desired trade-off between memory usage and capturing long-term dependencies, we might need to adjust the sequence length used as input to the model. Longer sequences can help capture more context but might require more memory. We need to normalize the input data before feeding it to the model. This can help improve convergence and make the training process more stable. Divide our

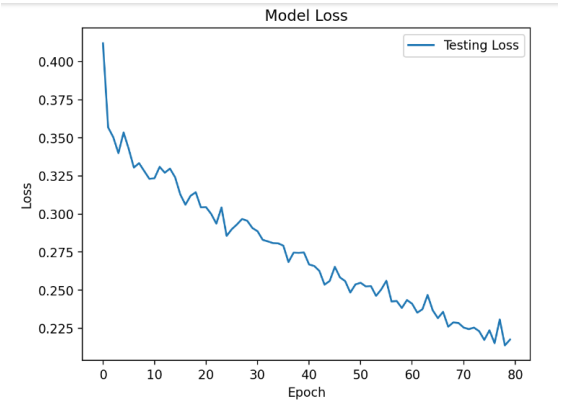
dataset into training, validation, and testing sets. This helps assess a model's performance on unseen data and prevents overfitting.

Training and Evaluation:

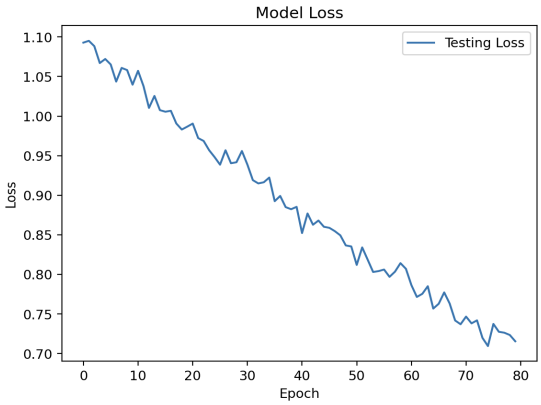
The Adamax optimizer is used to compile the model, and both loss and accuracy metrics are observed as it is being trained. Following training, the model is loaded for inference, and random sampling is used to create music using predicted probabilities. Finally, MIDI notes and chords are created from the produced music patterns and saved in a MIDI file.

This project makes use of the multi-class classification method sparse_categorical_crossentropy, which is appropriate for tasks where each class is represented by a single integer label. The accuracy statistic, which measures the percentage of properly predicted notes relative to the total number of predictions, is calculated by the model during training. Human evaluation is taken into account to improve the standard of music generated.

Results:



GRU Loss



LSTM Loss

Model	Accuracy	Testing Loss
LSTM	83%	0.71
GRU	91%	0.46

Based on the provided results, the GRU model performs better in terms of both accuracy and testing loss compared to the LSTM model. This suggests that the GRU architecture is more effective at capturing the underlying patterns in the music data and generating coherent music sequences.

Due to simpler architecture and fewer parameters, GRUs converge faster during training compared to LSTMs. This can save training time and resources. GRUs have a simpler architecture compared to LSTMs, with fewer gates. This simplicity makes it easier to train, especially when dealing with relatively smaller datasets.

In our project, we tried multiple learning rates and optimizers like Adam, Adamax, and RMSProp. But based on the results and quality of music generated, we found that in our model when we used a learning rate with 0.001 and Adamax optimizer, our model performed better when compared to other learning rates and optimizers.

Conclusion:

In conclusion, this project used LSTM and GRU networks to explore the world of artificial intelligence-produced music. We have shown that neural networks can understand complex musical patterns by analyzing MIDI files, generating input-output sequences, and applying good training. Although our computer-generated compositions have some promise, it is still difficult to replicate human creativity. This project not only demonstrates how technology and art can work together, but it also provides a basis for future research into the intersection of AI and music.

References:

1.How to Generate Music using a LSTM Neural Network in Keras" by Towards Data Science
2. "Music Generation using LSTM Neural Networks" by David Exiga
3. Eck, D., & Schmidhuber, J. (2002). Finding temporal structure in music: Blues improvisation with LSTM recurrent networks. Neural Networks, 14(6-7), 737-744.
4. Yang, Z., Cui, Z., Guo, M., & Yang, Z. (2017). MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation. In Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR).