Deep Learning – Based Melody Synthesis & Artistic Style Reproduction Using LSTM Networks

A Project Report

Submitted by

Group : 7

Batch-B

Antonio Roger P : CB.SC.U4AIE23104

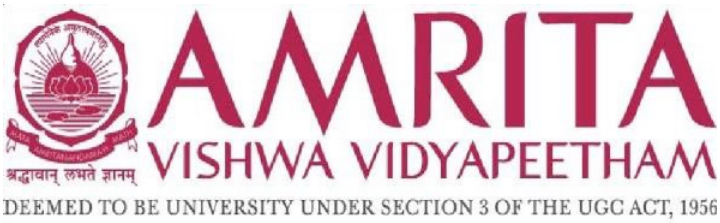
Adarsh P : CB.SC.U4AIE23109

B Viswesh : CB.SC.U4AIE23118

Naresh Kumar V : CB.SC.U4AIE23165

As a part of the subject

22AIE201 – FUNDAMENTALS OF AI



Department of Artificial Intelligence

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641112 (INDIA)

NOVEMBER 2024

AMRITA SCHOOL OF ENGINEERING
AMRITA VISHWA VIDYAPEETHAM
COIMBATORE - 641 112

DECLARATION

I hereby declare that the project work entitled **"Deep Learning-Based Melody Synthesis & Artistic Style Reproduction Using LSTM Networks"** submitted to Amrita Vishwa Vidyapeetham is a record of an original work done by me under the guidance of Asst. Prof. Dr.Abhishek S, Dean K.P. Soman, School of Artificial Intelligence. The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma. This project work is submitted in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Artificial Intelligence Engineering.

Place: Coimbatore
Date: **16-11-2024**

ACKNOWLEDGMENT

We wish to extend our sincere appreciation to Asst. Prof. Dr.Abhishek S, our supervisor, for providing outstanding guidance, insightful suggestions, constructive criticisms, and valuable input throughout this project.

Our heartfelt gratitude goes to Dean. K.P Soman, School of Artificial Intelligence for their valuable suggestions and encouragement at various stages of the project. I also express my thanks to all the members of the Department of Artificial Intelligence at Amrita Vishwa Vidyapeetham.

We express our deep gratitude to our colleagues for the continuous flow of ideas, as well as to my dear ones and all those who contributed to the successful completion of this project.

Amrita Vishwa Vidyapeetham

Contents

Abstract

Our project **"Deep Learning-Based Melody Synthesis & Artistic Style Reproduction Using LSTM Networks"** aims to generate classical music using LSTM , specifically through deep learning model trained on MIDI files of compositions by western composers. The key Objective is to take in a series of classical music compositions, process them to extract meaningful features, and use these features to train a model that can generate new, unique music that resembles the original style of the artist. The dataset utilized for this project consists of classical music MIDI files. Which includes compositions from various classical artists like Haydn, Mozart, and Beethoven. By pre-processing and feature extraction, the notes and chords from these MIDI files are extracted, cleaned, and stored for use in training the LSTM model.

Long Short-Term Memory (LSTM) network, it is a type of recurrent neural network (RNN) capable of learning sequential patterns. The model is trained to predict the next note or chord in a sequence based on the previous notes, using the extracted features from the MIDI files. The training dataset is split into sequences of notes, which are then used to train the model to predict the next element in the sequence. After training, the model is capable of generating new melodies by starting with a seed sequence as starting point and predicting subsequent notes iteratively. The output of the model is a MIDI file, which is a digital representation of the generated music, allowing for its playback.

The model successfully generates new melodies that resemble the style of the classical composers in the dataset. By further fine-tuning the model, experimenting with different dataset , and incorporating additional models , the quality of the generated compositions can be improved. This project highlights the potential of Deep Learning in the field of music composition, opening new possibilities.

Introduction

Music has always been an integral part of human culture, with its ability to evoke emotions, convey stories, and inspire creativity. In recent years, advancements in artificial intelligence have opened new possibilities in music generation and artistic style synthesis. Leveraging deep learning, particularly Long Short-Term Memory (LSTM) networks, has transformed the field of automated melody synthesis, allowing machines to not only mimic musical compositions but also generate novel sequences with stylistic elements akin to renowned composers.

Melody synthesis involves generating a sequence of musical notes to form coherent musical pieces, while artistic style reproduction aims to capture the nuances of an artist's signature style. In this project, we aim to develop a system that synthesizes melodies and reproduces artistic musical styles using LSTM networks trained on classical music datasets. The system is designed to analyse MIDI files, learn patterns, and generate new musical compositions that resemble the styles of famous classical composers such as Chopin, Mozart, Borodin, Schumann, Schubert, etc,.

Given the sequence-oriented nature of music, LSTM networks are particularly well-suited for this task. These networks excel at learning temporal dependencies, allowing them to predict and generate music based on previously learned patterns. By leveraging a large dataset of classical music, our system aims to create a model capable of producing new melodies while preserving the stylistic characteristics of the composers in the dataset.

Methodology

3.1 Dataset Collection and Preprocessing

The dataset used in this project is a collection of **classical music MIDI files** downloaded from Kaggle. The MIDI files are parsed using the music21 library, which extracts musical notes, chords, and other features.

The dataset is organized by composer, allowing for style-specific melody generation. Each composer's MIDI files are saved in a pickle format for efficient loading during training.

3.2 Data Extraction

A function named extract_notes extracts notes and chords from each MIDI file. The extracted data is stored in a list called Corpus, which serves as the foundation for model training.

The most frequent notes are identified using a **count dictionary**, and rare notes (those occurring less than a threshold number of times) are removed to focus the model on frequently occurring patterns.

3.3 Data Preparation for LSTM

The musical sequences are tokenized, converting notes and chords into numerical indices using a mapping dictionary.

The data is then split into input sequences (features) and corresponding target notes (targets). Each input sequence has a fixed length of 40 notes.

The input features are normalized, and the target notes are one-hot encoded to serve as the output labels.

3.4 Model Architecture

The model is implemented using a **Sequential LSTM network** with the following architecture:

Two LSTM layers: The first layer has 512 units with return_sequences=True to pass the output to the next LSTM layer. The second layer has 256 units.

Dropout layers are added to prevent overfitting.

A Dense layer with a softmax activation function serves as the output layer to predict the next note.

The model is compiled using the **Adamax optimizer** with a learning rate of 0.01 and trained using a categorical cross-entropy loss function.

3.5 Melody Generation

The trained model is used to generate new musical sequences by predicting the next note based on a seed sequence.

The generated notes are converted back into a MIDI file format using the chords_n_notes function.

Implementation

3.1 Environment Setup and Libraries

To start, we import several libraries essential for data processing, model building, and melody generation. TensorFlow and Keras are used for constructing and training the LSTM model, while other libraries like music21, numpy, and pandas are utilized for processing MIDI files and preparing the dataset. Additionally, matplotlib and seaborn are used for visualization.

```
import tensorflow
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adamax
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
import music21
from collections import Counter
import os
```

3.2 Data Loading and Preprocessing

We begin by loading MIDI files from a specified folder based on the user input. The files are parsed using the music21.converter module, which extracts the musical content.

```
composer_name = input("Enter the composer name: ")
filepath = os.path.join("/content/dataset/", f"{composer_name}/")

midis = []
for i in os.listdir(filepath):
    if i.endswith(".mid"):
        midi = music21.converter.parse(filepath + i)
        midis.append(midi)
```

3.3 Extracting Musical Notes and Chords

Once the MIDI files were loaded, we needed to extract individual notes and chords from them. The extract_notes() function handles this task by iterating through each MIDI file and identifying notes or chords.

```
def extract_notes(files):
    notes = []
    for midi in files:
        parts = instrument.partitionByInstrument(midi)
        for part in parts.parts:
            elements = part.recurse()
            for element in elements:
                if isinstance(element, note.Note):
                    notes.append(str(element.pitch))
                elif isinstance(element, chord.Chord):
                    notes.append(".".join(str(n) for n in element.normalOrder))
    return notes

Corpus = extract_notes(all_midis)
print("Total notes extracted:", len(Corpus))
```

The extracted notes were stored in the list Corpus, which served as our dataset for training the model.

3.4 Data Cleaning and Dictionary Creation

We remove rare notes that occur less than 100 times to reduce noise in our dataset. The cleaned list is then used to create mappings between notes and numerical indices.

```
count_num = Counter(Corpus)
rare_note = [key for key, value in count_num.items() if value < 100]
Corpus = [element for element in Corpus if element not in rare_note]

# Creating mappings for notes to integers
symb = sorted(list(set(Corpus)))
mapping = dict((c, i) for i, c in enumerate(symb))
reverse_mapping = dict((i, c) for i, c in enumerate(symb))
```

3.5 Data Preparation for Model Training

We split the cleaned Corpus into sequences of a fixed length (length = 40) to prepare inputs and targets for training the LSTM model. We reshape the input data and normalize it before applying one-hot encoding to the targets.

```
length = 40
features, targets = [], []

for i in range(0, len(Corpus) - length, 1):
    feature = Corpus[i:i + length]
    target = Corpus[i + length]
    features.append([mapping[j] for j in feature])
    targets.append(mapping[target])

X = np.reshape(features, (len(targets), length, 1)) / float(len(symb))
y = tensorflow.keras.utils.to_categorical(targets)
X_train, X_seed, y_train, y_seed = train_test_split(X, y, test_size=0.5, random_state=0)
```

3.6 Building the LSTM Model

We define an LSTM model using Keras with a sequential architecture. The model consists of two LSTM layers, each followed by a Dropout layer to prevent overfitting. Finally, a Dense layer with a softmax activation function is used for multi-class classification.

```
model = Sequential()
model.add(LSTM(512, input_shape=(X.shape[1], X.shape[2]), return_sequences=True))
model.add(Dropout(0.1))
model.add(LSTM(256))
model.add(Dense(256))
model.add(Dropout(0.1))
model.add(Dense(y.shape[1], activation='softmax'))

opt = Adamax(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=opt)
model.summary()
```

3.7 Training the Model

The model is trained for 200 epochs with a batch size of 512. The training history is stored to visualize the loss curve over epochs.

```
history = model.fit(X_train, y_train, batch_size=512, epochs=200)
```

We plot the training loss to monitor the learning process

```
history_df = pd.DataFrame(history.history)
fig = plt.figure(figsize=(15,4), )
fig.suptitle("Training Loss Vs Epochs")
pl=sns.lineplot(data=history_df["loss"])
pl.set(ylabel ="Training Loss")
pl.set(xlabel ="Epochs")
```

3.8 Model Validation

We split the training data further into training and validation sets to evaluate the model's performance. The accuracy of the model is measured on both sets.

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=0)

# Evaluate the model on the training and validation datasets
train_loss, train_accuracy = model.evaluate(X_train, y_train)
val_loss, val_accuracy = model.evaluate(X_val, y_val)

# Print training and validation accuracy results
print(f"Training Accuracy: {train_accuracy * 100:.5f}%")
print(f"Validation Accuracy: {val_accuracy * 100:.5f}%")
```

3.9 Melody Generation

We define functions to generate melodies based on the trained model. The chords_n_notes function converts sequences into MIDI format..

```
def chords_n_notes(Snippet):
    Melody = []
    offset = 0
    for i in Snippet:
        if "." in i or i.isdigit():
            chord_notes = i.split(".")
            notes = [music21.note.Note(int(j)) for j in chord_notes]
            chord_snip = music21.chord.Chord(notes)
            chord_snip.offset = offset
            Melody.append(chord_snip)
        else:
            note_snip = music21.note.Note(i)
            note_snip.offset = offset
            Melody.append(note_snip)
        offset += 1
    return stream.Stream(Melody)
```

3.10 Melody Generation

After training, the model can generate new melodies based on a seed sequence. The Melody_Generator() function uses the trained model to predict subsequent notes based on a given seed, allowing the generation of music.

The diversity parameter is used to control randomness in predictions, creating more varied melodies.

The generated notes are then converted back into a music21 stream to save as a MIDI file.

```
def Malody_Generator(Note_Count):
    seed = X_seed[np.random.randint(0, len(X_seed) - 1)]
    Notes_Generated = []
    for i in range(Note_Count):
        seed = seed.reshape(1, length, 1)
        prediction = model.predict(seed, verbose=0)[0]
        index = np.argmax(prediction)
        Notes_Generated.append(index)
        seed = np.append(seed[0], index / float(L_symb))[1:]

    Music = [reverse_mapping[char] for char in Notes_Generated]
    Melody = chords_n_notes(Music)
    return Music, Melody

# Generate a sample melody
Music_notes, Melody = Malody_Generator(100)
Melody.write('midi', 'Melody_Generated.mid')
```

This section covered the **step-by-step implementation** of a deep learning-based melody generation system. The key components included:

1. Loading and parsing MIDI data.
2. Preprocessing notes and filtering rare ones.
3. Building and training an LSTM model.
4. Using the model to generate new musical sequences.

Results

Figure 1: *Code Execution for MIDI File Generation*

This figure shows the Python code snippet used to generate and save a MIDI file.
The music21 library's Melody.write method is employed to write the generated melody to a file named Melody_Generated.mid. The success of this operation is confirmed by the return of the filename.

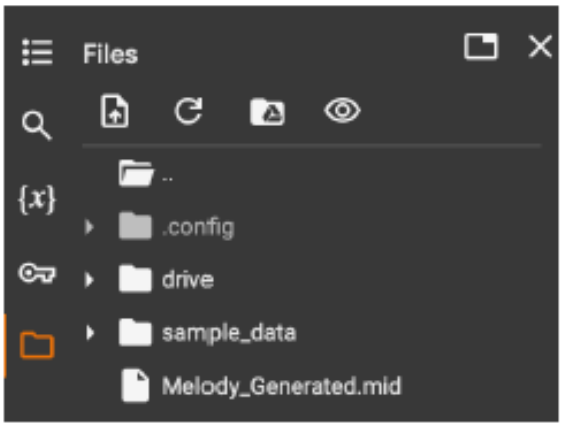


Figure 2: *File Explorer Displaying the Generated MIDI File*

The generated MIDI file (Melody_Generated.mid) is displayed in the file explorer within the working directory of the environment. This confirms that the file has been successfully created and saved.

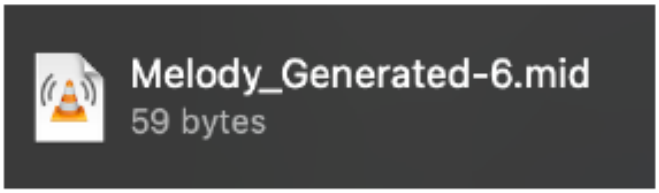


Figure 3: *Generated MIDI File Ready for Playback*

Melody_Generated.mid file as it appears in the environment, ready for playback. indicating that the melody generation process has successfully produced a file of .mid format.

CONCLUSION

The project, *"Deep Learning-Based Melody Synthesis & Artistic Style Reproduction Using LSTM Networks"*, demonstrates the potential of using advanced machine learning techniques to generate music and replicate the artistic styles of renowned composers. By employing Long Short-Term Memory (LSTM) networks, we successfully modeled the sequential nature of music and produced melodies that capture stylistic elements from classical composers. The key outcomes of this project include:

1. The development of a robust preprocessing pipeline to handle classical music MIDI datasets.
2. Implementation of an LSTM-based neural network that effectively learns temporal dependencies in musical sequences.
3. Generation of new musical compositions that reflect the styles of the composers in the dataset, validated by producing playable MIDI files.

Future Work

1. **Dataset Expansion:** Incorporating more comprehensive datasets with compositions across diverse genres and composers to improve the model's learning.
2. **Advanced Architectures:** Exploring other deep learning models, such as Transformers or Variational Autoencoders (VAEs), to enhance the quality and diversity of the generated compositions.
3. **Real-Time Generation:** Developing tools for real-time melody generation to broaden the application in interactive and live music systems.
4. **User Interactivity:** Allowing user-defined parameters, such as mood or tempo, to guide melody synthesis, making the system more versatile and user-friendly.

This project underscores the potential for blending artificial intelligence with creative disciplines, paving the way for innovative applications in music composition and beyond.