Key Consistent Tune Generation using Sequence Modelling

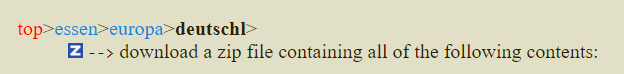
# Project Description

# This project is a sophisticated music generation system that transforms Kern files into melodious compositions using advanced deep learning techniques. Leveraging the power of the music21 library for musicological analysis and preprocessing, the system employs three distinct types of Recurrent Neural Networks (RNNs)—LSTM, GRU, and Vanilla RNN—to learn and generate musical sequences. The resulting compositions are exported as MIDI files, offering a versatile format that can be easily played and appreciated using applications like MuseScore. Whether you're exploring the frontiers of computational creativity or seeking new musical inspirations, this project bridges the gap between data-driven algorithms and harmonious artistry.

# Installation

1. Open google colab and upload the code provided here : [Project\_DL\_Seuence\_Modelling\_RNN.ipynb](https://github.com/AnanyaShankarChittoor/SequenceModellingMusic/blob/main/Project_DL_Seuence_Modelling_RNN.ipynb)
2. Download dataset from here: <https://kern.humdrum.org/cgi-bin/browse?l=essen/europa/deutschl>

Click on the Z icon



1. Upload dataset to google drive
2. Run the google colab code.Make sure you change runtime to GPU.
3. Your model will get trained on “Ballad” subdirectory of the dataset. You can change it to a different one if needed
4. Once the whole code completed running three different .h5 files will be created .
5. Download them and keep them ready. Also download the mapping\_config.json and tracks\_combined\_dataset files.
6. Go to this code. You can set it up locally on any IDE. I used VS code as that is what I am most comfortable with: [MusicFolder](https://github.com/AnanyaShankarChittoor/SequenceModellingMusic/tree/b97afb580454436643a1af3c54631ce89ee4b90e/Music)
7. Install tensorflow and music21 if not already installed
8. Select python interpreter in VSCode and setup interpreter:
9. Go to terminal and execute:
   1. pip install --upgrade pip setuptools
   2. pip install tensorflow
   3. pip install numpy
   4. pip install music21
10. Move the .h5 ,mapping,config and tracks\_combined\_dataset to the Music Folder
11. Run melodyGenerator.py.
12. Download museScore.
    1. URL <https://musescore.org/en>
13. Open the midi file using museScore to listen to the melody generated. Use the “Open ” option shown in screenshot and load the midi file to listen to the song.

A screenshot of a computer program

Description automatically generated

# How the code works

## Project\_DL\_Seuence\_Modelling\_RNN.ipynb

1. Initially we load all songs from the dataset. They are present in kern format.
2. Once loaded we go through each one of them and start preprocessing:
   1. Flatten the song
   2. Filter out all objects which are not notes and rests
   3. Filter out song based on duration (to simplify song for DL model). Also most songs will have usually the accepted durations given
   4. We will be shifting everything to A minor or C major. This makes it easier for the model to learn. If not this model will have to learn all 24 keys. Key shifting:
      1. Get the key from song. Usually stored in the first measure.
      2. If key is not given in the song then estimate it using m21 library
      3. If the song is in “**Major**” scale create an interval object which gets pitch of current song and pitch of C major
      4. If the song is in “**Minor**” scale create an interval object which gets pitch of current song and pitch of A minor
      5. Shift key using this interval value
   5. After this we have encoded the whole song to its midi version and save it. We have handled rests as well.
   6. Once the above step is done for all songs we create a combined file. We use a separator (/ 64 times) to separate different songs.
   7. The next step is to create a mapping. This contains the vocabulary of the songs which is input. Vocabulary refers to all symbols in the dataset. Each symbol will have a representation. Eg: 64,68,\_ etc these will be mapped to a number using for loop. It is like a lookup table.
   8. We need to next create training sequences. This is basically loading all the songs mapping them to integers and then one hot encode them.S
3. Once we have done preprocessing we will create 3 basic models as given in the code. Once it is trained .h5 files are created. You can also view the training accuracy and loss in the end part of the code. I have attached the same in PPT.

## melodyGenerator.py

1. Here first we need to give the code the .h5 I,e weights of the model.
2. For each model we get the mapping dictionary.
3. We need to provide the code a seed value . It can be whatever you want but correspond to its midi note number.
4. We one hot encode the seed value and predict using the model the tune.
5. We also make use of something called temperature. This is a value used to create randomness.
6. Once all of this is done we create midi file using music21 Stream() function.

# Experiments and Results

1. **Experimenting with Hyperparameter Tuning**:
   * **Objective**: We experimented with different learning rates on the three RNNs. The most optimal outcome in the least amount of time was received in 0.001 LR but for the vanilla RNN it didn’t converge hence we reduced LR to 0.0001. We also experimented with tanh and sigmoid activation functions. But since the model’s output varies and isn’t just a single class it was not the optimal choice.
   * **Result**: Despite extensive tuning, including grid search and random search methods, the improvements in melody quality were marginal. Some configurations even led to overfitting, causing the model to generate repetitive and uninspired melodies.
2. **Incorporating Attention Mechanisms**:
   * **Objective**: To enhance the model’s focus on important parts of the sequence by integrating an attention layer into the RNN architecture.
   * **Result**: Although attention mechanisms are powerful in many NLP tasks, they did not translate well to this music generation task. The generated melodies were more disjointed, with abrupt transitions that made the output sound very weird
3. **Switching to CNN-LSTM Hybrids**:
   * **Objective**: To combine convolutional layers with LSTM layers to capture local patterns in the music before passing the sequence to the LSTM.
   * **Result**: The hybrid model was overly complex and did not perform better than the pure LSTM models Also I was struggling to understand the structure. The convolutional layers added unnecessary complexity.

# Known Issues

* This code will work for only a small number of training data as Google colab crashes. If you need to train it on the whole dataset you would need a strong RAM and GPU.
* Occasionally, the models may produce monotonous or repetitive sequences, especially when the temperature parameter is not carefully tuned during melody generation. This can result in not so good songs/tunes
* The output melodies are highly sensitive to the temperature setting used during generation. Low temperatures may result in overly conservative and repetitive melodies, while high temperatures can lead to chaotic and incoherent sequences. Hence if you play around with that value be really careful.
* The current implementation might be primarily focused on generating monophonic melodies (single-note sequences). Extending the system to support polyphonic compositions (multiple simultaneous notes) could introduce additional complexity and challenges. Also it needs more music theory knowledge which I currently lack.

# Future Roadmap

In future iterations of this project, we plan to explore and integrate alternative and advanced architectures for time series prediction and sequence generation. Potential technologies include:

* **Transformers**: Originally designed for natural language processing, transformers have shown great promise in handling sequence data, offering parallelization benefits and capturing long-range dependencies more effectively than traditional RNNs.
* **Temporal Convolutional Networks (TCNs)**: TCNs are a type of convolutional neural network designed to process sequential data. They provide the advantages of faster training times and a more stable training process compared to RNNs.
* **WaveNet**: Developed by DeepMind, WaveNet is a deep generative model for producing raw audio waveforms. It has the potential to be adapted for musical sequence generation, offering high-quality outputs.
* **Attention Mechanisms**: Attention-based models, including those incorporated within transformers, allow the model to focus on specific parts of the input sequence. This can improve the generation of more contextually relevant music.
* **Variational Autoencoders (VAEs)**: VAEs, when combined with RNNs, can be used for generating sequences in a more controlled manner, potentially improving the creativity and diversity of the generated melodies.

By exploring these technologies, we aim to enhance the quality, variety, and creativity of the music generated, pushing the boundaries of what is possible in computational music composition.

# References

* <https://kern.humdrum.org/cgi-bin/browse?l=essen/europa/deutschl>
* <https://musescore.org/en>  
  <https://www.youtube.com/MuseScore>
* <https://www.kaggle.com/datasets/sebastianeck/essen-folksong-database-conversion-and-tokenization>
* <https://github.com/CellMigStandOrg/essen-dataset-analysis>