AC-10 – From Code to Chords: Unleashing AI in Music

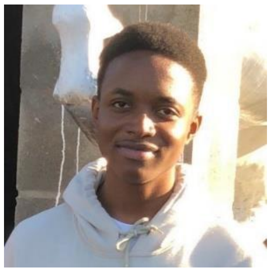
Final Draft

CS 4850 – Sharon Perry

Nov 16, 2024 – Fall '24

Website: https://ac-10-research.github.io/ac10.github.io/index.html

GitHub: https://github.com/AC-10-Research

Joseph Zboinski	 Michael Egwuatu
Selam Kelil (Team Leader)	Trey Wilson

Team Members:

Name	Role	Cell Phone / Alt Email
Joseph Zboinski	Developer	919.912.6612 mjzboinski@hotmail.com
Selam Kelil (Team Lead)	Developer/ Project Manager	selamkelil@gmail.com
Trey Wilson	Developer/Documentation	478.508.8746 tjwilson003@gmail.com
Michael Egwuatu	Documentation	678.956.3992 mikegwuatu@gmail.com
Arthur Choi	Project Owner/Advisor	achoi13@kennesaw.edu

Table of Contents

1. Introduction

The project applies machine learning to music pattern recognition and composition, simulating human-like learning in music creation. Unlike traditional audio methods, it uses MIDI (Musical Instrument Digital Interface) data to capture musical patterns and timing, like sheet music. By training on these patterns, the model mirrors human processes in composition. The project concludes with a web-based demo showcasing AI's potential in music generation and recognition.

1.1. Motivation

The primary motivation for this project is to investigate whether a machine learning model can learn music similarly to how children do—either through abstraction (pattern recognition) or memorization of specific patterns. Using musical scales as the basis for training, the model will study recurring patterns within these structures. This approach will help us assess which learning method—pattern recognition by abstraction or direct memorization—might allow for greater creativity in music composition. Understanding the model's ability to emulate human-like creativity in learning music scales could reveal valuable insights into AI's potential for artistic expression and innovation.

1.2. Project Goals and Scope

The project encompasses several key objectives:

- Testing current dataset and model capabilities
- Construction of a testbed for training Neural Networks
- Creation and testing of a new model using different scales
- Development of a new tutorial on Jupyter notebook
- Training a model on specific genres and testing on different music for genre identification

The system developed in this project comprises two main components:

- A music-generating recurrent neural network (RNN) implemented in Python
- An online educational tutorial created using Jupyter Notebook

The RNN is trained on various music scale patterns to learn and predict/generate musical patterns, with the capability to both reproduce existing compositions and create original pieces. The Jupyter Notebook tutorial guides users through the process of training their own music-generating models, providing a hands-on learning experience.

2. Requirements

2.1. System Description

The system generates predict scales that can create a sensible musical composition. The music synthesizing model may be integrated into different systems as determined by project requirements.

2.2. Environment

The development environment includes:

- Environment: The system uses Python, TensorFlow for RNN development, and Jupyter Notebook for the tutorial and interactive learning.
- Data Format: MIDI files were selected for their ability to store detailed musical patterns (notes, timing) in a way like sheet music. Major and minor scales were used for providing more structured data for the model learning.

2.3. Functional Requirements:

- Music Generation: Enable music synthesis using an RNN trained on MIDI data.
- Tutorial: Provide a comprehensive guide on training a music-generating model in Jupyter Notebook.
- Website: To display the results of the project.

2.4. Non-functional Requirements:

- Output Quality: Ensure high quality and realism in generated music, minimizing noise.
- Performance: Optimize model to maintain consistent output quality and efficiency.
- Ethics and Privacy: Adhere to data privacy guidelines, using only publicly available or licensed datasets.

2.5. Technical Constraints:

- Limitations

2.6. Performance Requirements

The system must maintain:

- Consistent output quality
- Effective noise management
- Predictable note generation

- Efficient resource utilization

2.7. Data Requirements

Data handling encompasses:

- Base song creation in MIDI format
- Dataset expansion through song repetition
- Training data generation through sequence splitting
- Input sequence formatting for RNN compatibility

3. Analysis

3.1. System Analysis:

- Environment: The system uses Python, TensorFlow for RNN development, and Jupyter Notebook for the tutorial and interactive learning.
- Data Format: MIDI files were selected for their ability to store detailed musical patterns (notes, timing) in a way like sheet music. Major and minor scales were used for providing more structured data for the model learning.

3.2. Challenges:

- Resource Constraints: Training deep neural networks requires extensive computational power, especially for high-dimensional MIDI data.
- Noise Reduction: Ensuring clarity in generated music output to avoid distracting noise or unwanted artifacts.
- Consistency in Music Quality: Maintaining uniform quality across different generated compositions.

4. Design

A modular structure was used in the architecture separating the music-generation model from the educational tutorial, enabling independent development and testing of each module. The system architecture comprises two primary sub-systems:

4.1. Music Generation

The music generation module handles:

- Model Training Module: Trains the RNN to predict musical sequences, with hyperparameter tuning (NN, epochs, randomization, repetition …) and evaluation tools.
- Music Synthesis Module: Generates music by using the trained model to predict sequences.
- Training and deployment of neural networks for music generation
- Data preprocessing (scale preparation) and creation of midi-files using scale patterns
- Model training and optimization
- Performance evaluation metrics
- Visualization of the network's learning process
- Music synthesis and output generation (Futuristic Goal)

4.2. Tutorial

The Jupyter notebook interface provides tutorial provides interactive learning and experimentation for users, with explanations, code examples, and widgets for real-time adjustments.

- Educational interface for user interaction
- Instructional content and code examples
- Interactive widgets for experimentation
- Guidance through environment setup
- Training procedures and best practices
- Theoretical foundations of neural networks and music generation

5. Development Process

The development process encompasses data preparation, architecture implementation, and training procedures. Each phase is structured to ensure optimal system performance and maintainability through systematic data handling, model construction, and training optimization.

5.1. Data Preparation

- Song Definition: Create a base song in sequence of numbers (scale format), which is then repeated to expand the dataset
- Dataset Expansion: Repeat the base song multiple times to create a sufficiently large dataset for training

- Data Formatting: Segment the repeated song into input and target sequences, structured for compatibility with the RNN model

5.2. Model Development

- Model Definition: An RNN model with specified number of neurons and dense layers for predicting the next note in a sequence
- Compilation and Training: Model is compiled with Mean Squared Error (MSE) loss and optimized using the Adam optimizer, with early stopping employed to prevent overfitting during training

6. Test Plan, Performance Analysis, and Report

The testing phase implements an evaluation framework to assess model functionality, performance, and generalization capabilities across various musical contexts, while documenting results for analysis and optimization.

6.1. Test Requirements

- Core Functionality: Scale → MIDI conversion, song creation, note prediction, tutorial functionality, music synthesis
- Model Evaluation: Prediction accuracy, neuron size comparisons

6.2. Testing Plan:

- Accuracy Testing: Test model on transposed versions of the song to assess generalization
- Neuron Size Comparison: Compare model performance across various neuron sizes for optimal configuration
- Prediction Accuracy: Evaluate accuracy of predicted notes against target notes from input song

6.3. Results:

- Model Accuracy: Predictions showed consistent accuracy across different transpositions
- Training Loss: Loss per epoch plotted to visualize training progress and evaluate stopping criteria
- Performance Charts: Charts of model accuracy for each transposition created, highlighting model's robustness.

7. Version Control

Version control is managed through GitHub under the AC-10-Research . The repositories are hosted on the organizational GitHub account, where team members work in separate files then push. The final version of the code was placed in a v2 folder.

8. Summary

This project explored the intersection of AI and music by developing a system that can generate music through machine learning. With a focus on simple musical scale. It included an RNN model trained on different data and an educational tutorial. Challenges like limited knowledge among teammates about music theory and application of machine learning provided a learning opportunity. The progress in the project was made with room for future refinement in creating improved model with genre classification and music synthesis.

9. Appendix

9.1. Glossary and Abbreviations

- ML – Machine Learning
- MIDI – Musical Instrument Digital Interface
- NN – Neural Network
- RNN – Recurrent Neural Network

9.2. References

- TensorFlow Tutorial: Generate music with an RNN | TensorFlow Core