MIDI Music Generator via Adversarial Inverse Reinforcement Learning

DRAFT

Pranav Banuru
pbanuru@uci.edu

Jun Yan
yanj16@uci.edu

CompSci 175: Project in Artificial Intelligence
University of California, Irvine
June 2023

1 Introduction and Problem Statement

The intersection of artificial intelligence and music presents an intriguing frontier, where the creation of music is no longer solely a product of human creativity but can be facilitated by computational means. While existing algorithms can generate music based on predefined rules and patterns, the challenge lies in creating compositions that encapsulate the emotional depth, creativity, and structural nuances inherent in human-composed music. Our project aimed to address this challenge by developing an AI agent capable of predicting the next chord or note in a composition, with the ultimate goal of generating a complete song.

To achieve this, we trained our model on a genre-specific dataset of MIDI data, a file format that encapsulates information about the notes, timing, velocity, and other musical attributes of a song, without the actual audio data. We employed a method known as Adversarial Inverse Reinforcement Learning (AIRL), which allowed us to evaluate the quality of a chord or note in relation to the preceding music through a generated score. This score, generated via a learned grading system, was used instead of traditional music theory rules, which can be hard to define and often fall short in capturing the complexity, diversity, and subjectivity of music, especially across different genres. By leveraging this scoring system, we aimed to create an AI agent able to learn, understand, and ultimately create music. However, our agent did not fully succeed in this task, indicating that further work is required.

2 Related Work

In the realm of music generation, various methods have been employed to tackle the challenge. Google Magenta, for instance, utilized Supervised Learning based Long Short-Term Memory (LSTM) networks to generate single note melodies via their MelodyRNN model, leveraging a vast corpus of music data for training. Daphné Lafleur at the Université de Montréal adopted a different approach, employing Deep Q-Learning (DQN) with Constraint Programming, guided by music theory rules, and reinforcement learning for next-note generation. Sam Lowe at the University of North Carolina at Chapel Hill, on the other hand, used Generative Adversarial Imitation Learning (GAIL), a different Inverse Reinforcement Learning Approach that doesn't technically involve reinforcement learning, as no reward function is produced. In comparison to GAIL, AIRL is better at adapting to unseen environments, which motivated our use of AIRL.

A paper titled "Polyphonic Music Composition: An Adversarial Inverse Reinforcement Learning Approach," currently under review by anonymous authors, bears the closest resemblance to our

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

project. It aims to craft a reward function and create polyphonic (multi-note) music, diverging from the single-note focus of previous works. Both this paper and Lafleur's project integrate music theory rules, contrasting with our approach, which seeks to learn solely via the learned reward function. The anonymous authors' paper combines the learned reward function with music theory rules to train the agent and employs a Bi-Axial LSTM model, which is better tuned to music applications specifically, as opposed to a regular LSTM model. Regrettably, we discovered this paper too late in our project timeline, preventing us from building upon their work. Consequently, we started from scratch.

3 Data Sets

We use a curated dataset of MIDI files, utilized as the main training data for our music generation model. MIDI (Musical Instrument Digital Interface), a standardized protocol for conveying musical information, encompasses specifics such as notes, chords, and tempo. The structured and detailed nature of MIDI files makes them aptly suited for machine learning applications in music.

This MIDI dataset is diverse, encompassing a wide variety of musical compositions across a particular genre with varied styles. Each MIDI file contains multiple tracks, with each track representing a unique instrument or voice. In processing these files, we extracted the sequences of notes and chords while consciously omitting percussion tracks, given their limited contribution to melodic and harmonic aspects.

Overview of our MIDI dataset: Data Set Statistic Number of Songs: 92 Genre: Final Fantasy / Anime Style Music Link: https://github.com/Skuldur/Classical-Piano-Composer/tree/master/midi$_songs$

The table provides key insights into the MIDI dataset which consists of 92 files covering distinctive genres like Final Fantasy and Anime/Video-Game Style Music. By utilizing this eclectic assortment of MIDI files, our intent is to immerse the model in an extensive range of musical patterns, harmonies, and structures within a specific context. This exposure aids the model in deciphering inherent patterns and relationships between musical elements, enabling it to generate compositions that reflect the essence of the training data.

Moreover, the AIRL (Adversarial Inverse Reinforcement Learning) approach we employed allows the model to adapt to various musical styles within the data. This adaptability is pivotal when generating compositions in genres beyond the ones in the training data, offering a robustness that is crucial for real-world applications.

The MIDI dataset's inherent flexibility and scalability provide room for future growth. By integrating additional MIDI files from diverse sources and genres, we can further broaden the model's understanding and capability to generate music that resonates with a wider range of styles and preferences. Such enhancements to the dataset underline our aspiration to further improve the model's adaptability to genres outside its training scope, thereby enhancing the universality and applicability of our approach, beyond reward function adaptability.

4 Description of Technical Approach

4.1 Data Preprocessing and Representation

The first step is preprocessing the MIDI data. We use the Music21 library to convert each MIDI song in our training set into a Python-compatible representation. For each song, we maintain information about the offset, duration, and pitches of each chord. We use the term "chord" to describe both single and multi-note events. For simplicity, we do not maintain note velocities, but this could be added later.

We also calculate the rest, or the time between the current and previous note events (current_offset - prev_offset), and store this information. Music is highly dependent on relative offset (or rest), so we include it as part of the training. Additionally, we include the key of the song, as adjacent musical keys often share many of the same notes, and we want to capture the relationship of parallel keys (major and minor keys that share the same notes but use a different ordering). For simplicity, we only work with major and minor modes.

To encode the key, we use a circular encoding, treating the Circle of Fifths like a unit circle and using sine and cosine to find the position on the unit circle with two numbers. We also use -1 for minor and +1 for major modes to store the key in three values. The data is then standardized to have a mean of 0 and a standard deviation of 1, to keep data within a soft range easier for machine learning.

The pitches of each chord are represented as a multi-hot binary vector of size 128, where the index is 1 if that note is active, and 0 otherwise (MIDI notes can be represented from 0 to 127). Songs are now represented as sequences of chords, each with 134-length vectors containing [keyX, keyY, mode, offset, rest, duration, midi0active, midi1active, ..., midi127active]. We then place all songs back to back and collect sequences of 30 chords, along with the next chord as the expected output (e.g., chords 1-30 with chord 31, chords 2-31 with chord 32, etc.). For the expected output, we remove the key information and the offset, resulting in a vector of size 130.

4.2 Adversarial Inverse Reinforcement Learning

Our primary method is Adversarial Inverse Reinforcement Learning (AIRL), which uses two models trained against each other: the policy model, which predicts the next action, and the discriminator model, which tries to distinguish between what the policy has generated and our expert trajectories (the sequences and expected outputs).

4.2.1 Policy Model

For the policy model, we use an LSTM because it works well with sequential data like music, capturing both long and short-term patterns. The LSTM takes in the (30, 134) state sequence and outputs the (130,) action.

4.2.2 Discriminator Model

For the discriminator model, we use a regular neural network with two linear layers and ReLU activation. It takes in a (30*134+130,) vector (concatenating the state and action) and outputs a single value via a sigmoid function, indicating the likelihood of our input being part of an expert trajectory over the policy's action. Our intent was to use this discriminator's output as part of our reward function. However, due to unsatisfactory performance of our policy model, as described in the Experiments and Evaluation section, we didn't get that far.

5 Software

In our project, we utilized Python alongside a host of other libraries to accomplish various tasks:

- music21: This library facilitated handling MIDI data and analyzing the key of songs.
- pickle: We used this for storing and loading processed data, enhancing startup times in the event of a kernel crash.
- google.colab.drive: This was essential in storing pickled data, model checkpoints, and our standardization scalers for testing.
- numpy and pytorch: These were pivotal for data handling and for the development and training of our LSTM/Discriminator models.
- tqdm: This was helpful for visualizing the remaining time in data processing.
- matplotlib: We made attempts to use this for visualizing training progress, but were unsuccessful.

In terms of code we wrote ourselves, we implemented several features:

- Data standardization: We created a subsection for standardizing song data, converting pitch to multihot vectors, flattening chords, creating sequences, and generating output data.
- Model definitions: We defined our LSTMModel and Discriminator classes for the policy and discriminator models. These were used in the AIRL training loop, where the policy model generates actions (music sequences) and the discriminator model distinguishes between the generated sequences and real data.
- Training process: We created a process to train the models that involved steps such as splitting the data into training and validation sets, training the policy and discriminator models, calculating the losses, and updating the models based on the losses. The training process continues for a specified number of epochs or until the validation loss does not improve for a certain number of epochs (determined by the patience parameter). We also implemented code for tracking the training and validation losses over time.

- AIRL implementation: We attempted to use the preexisting imitation library to manage our AIRL training. However, due to compatibility issues, we ended up implementing the AIRL from scratch.

We also employed Google Colab Pro for model training due to the high RAM requirements. As a result of implementing the AIRL ourselves, we didn't need the custom gym environment we initially constructed, nor the imitation library. There are still some unresolved issues with the imitation library, which we plan to investigate further.

6 Experiments and Evaluation

During our project, we utilized an LSTM model to train on processed MIDI data with the goal of generating novel musical compositions. We employed an adversarial learning framework, with the Adam optimizer being used to train both the LSTM policy model and the discriminator model. The loss function was formulated as the negative mean reward. Specifically, the policy loss was set as the negative mean of the rewards given by the discriminator model for the actions generated by the policy model (policy_loss = -torch.mean(policy_rewards)). Simultaneously, the discriminator loss was computed as the difference between the negative mean of the rewards for the expert actions and the mean of the rewards for the policy actions (discriminator_loss = -torch.mean(expert_rewards) + torch.mean(policy_rewards)).

Our training data consisted of sequences of chords extracted from MIDI files. Expert-generated sequences served as the reference labels. The LSTM model was trained to generate sequences that the discriminator model would classify as expert-generated. On the other hand, the discriminator model was trained to differentiate between expert-generated and LSTM-generated sequences.

We performed the training over a predefined number of epochs, using an early stopping mechanism to avoid overfitting. We monitored the validation loss, which was obtained from a data split of 20% validation and 80% training. If the validation loss did not show improvement over a certain number of epochs (set by the early stopping 'patience'), we halted the training.

The training was planned for 500 epochs with early stopping patience set at 75 epochs. However, the early stopping mechanism triggered after 178 epochs due to no further improvement in the validation loss. At this point, the policy loss was -1.0, suggesting that the policy network was generating actions almost indistinguishable from the expert actions in the eyes of the discriminator. The discriminator loss was around 2.497e-0, implying that the discriminator was also performing reasonably well in distinguishing between the expert actions and the policy network's generated actions.

Despite our efforts, we faced difficulties in visualizing the training progress of both the policy and discriminator models due to issues integrating Matplotlib with Pytorch.

For the evaluation phase, we provided a 30 chord sequence from a new song the model had never seen before and then denormalized the output chord. The results, rest and duration, respectively, were -5.2717485 and -9.251137. The rest and duration were negative, and the duration was exceedingly long, indicating our model did not successfully converge, and did not learn to produce rational values. These results suggested that further optimizations and adjustments to the model architecture will be necessary for improved performance. Given the lack of policy performance, it suggests the discriminator is also lacking, and is not quite ready to train an RL agent from scratch in a new environment.

7 Discussion and Conclusion

The pursuit of developing a LSTM model for music generation has been insightful. We've navigated the intricacies of data preprocessing, music theory, and machine learning applications - an experience that emphasized the importance of understanding our data at a granular level, specifically MIDI files.

Our venture into adversarial learning frameworks, where we trained a policy model and a discriminator model simultaneously, was equally revealing. While our policy model successfully generated actions classified as expert-like, the results were not entirely as expected. This highlighted the complexity of adversarial frameworks and the challenge of interpreting loss metrics.

While working with StableBaselines3 and the imitation library, we faced hurdles due to limited online resources and documentation, indicating a need for better community support in AI research.

Despite these challenges and the constraints of our computational resources which necessitated a scaled-down model, we made meaningful progress. We view these outcomes as stepping stones towards the goal of AI-based music composition.

For future research, we aim to:

- Delve deeper into the successful methodologies used by the Unnamed Authors, potentially gaining insights to enhance our model's performance.
- Improve user engagement by allowing real-time interaction and personalization.
- Explore and test more complex models as computational resources allow.
- Consult with professionals to refine our approach and address encountered challenges more effectively.

In summary, our project represents a valuable stride towards AI-enabled music composition. While we encountered several obstacles, and our outcomes didn't align entirely with our expectations, the experience and insights gained have fuelled our optimism for future work. With further refinement and exploration, AI-generated music has the potential to assist musicians, inspire creativity, and contribute to the evolution of musical expression. We are excited to explore this realm further.