An Inverse Reinforcement Learning Approach to Generative Music

By
Sam Lowe

Senior Honors Thesis
Computer Science
University of North Carolina at Chapel Hill

April 9, 2020

Approved:

Dr. Ming Lin, Thesis Advisor

Dr. Dinesh Manocha, Reader

Dr. Don Porter, Reader

ABSTRACT

In this work, we explore the applicability of reinforcement learning (RL) techniques to generative music, with a particular focus on interactive musical agents. Despite the popularity of both generative music and reinforcement learning as topics of study, only a limited amount of research has explored their intersection, which we posit is partially due to the difficulty of specifying a reward function to describe musical behavior. However, recent advances in inverse reinforcement learning have resulted in algorithms that can train agents based on expert data in absence of a reward function. Building off these successes, we present *Melodic Imitator*, a system that learns to generate melodies using inverse reinforcement learning. We utilize an inverse RL algorithm, generative adversarial imitation learning (GAIL), which trains a pair of neural networks to generate trajectories that match the occupancy measures of the expert data set with only a few training examples. The sample efficiency of this approach means that it can potentially be used to create improvisational music companions, agents that learn to imitate the particular style of a musician, something that has been a research ideal since the early 2000s. We present several experiments in which we vary our data representations in order to compare the musical richness of the content generated in response. Finally, we describe a live demonstration system that adapts our system to a live musical context.

ACKNOWLEDGEMENTS

I must begin by expressing my deep gratitude to Professor Lin for all of her help and support over the past four years and especially for serving as the advisor to this project. I also want to acknowledge Professor Manocha and Professor Porter for serving as my second and third readers. Finally, I would like to thank Professor Chelsea Finn at Stanford University for her guidance on the algorithms which might hold the most promise for my applications. I hope these results speak to the great generosity of all the aforementioned.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

The application of artificial intelligence algorithms for the purpose of generating novel musical content has been the subject of ongoing research since as early as 1957, when Hiller and Isaacson created the first completely AI-generated score [1]. Their work, and the work of many musical AI researchers since, represents one particular direction for a much broader interest in the AI community of exploring the ability of artificial intelligences to behave creatively. Given that creativity is a central component in human intelligence, it seems logical that a truly intelligent artificial system will also be able to exhibit creative behavior, meaning research into machine creativity is of importance to the entire field of AI [2].

1.1 Motivation

Since developments in generative music carry such great consequence for AI efforts in general, it is perhaps surprising that the branch of reinforcement learning has received so little attention across these works. Recent successes in inverse reinforcement learning, in particular, provide strong indications of the potential of such approaches for applications to generative music; these methods are highly sample-efficient and alleviate some of the difficulties facing previous researchers investigating the intersection of reinforcement learning and music, namely the specification of a reward function. The potential of this unexplored avenue is the main motivator for our work and experiments on the applicability of such approaches.

1.1.1 Interactive Musical Agents

Many recent advances in reinforcement learning have given particular focus to the development of systems that can operate in real-time environments. The overlap of

deep learning and reinforcement learning has been especially fruitful in this endeavor, as many deep models are capable of performing real-time calculations once trained. This points to a potential expansion of efforts to use reinforcement learning for generative music into the area of interactive musical agents, live performance-based systems that process and generate musical content in real-time. Investigating the feasibility of building such a system with reinforcement learning methods was a core goal of our research.

1.2 Literature Review

We now discuss previous work in the research areas of consequence to our research.

1.2.1 Generative Music

In the time since Hiller and Isaacson unveiled the music of their rule-based system, many projects have sought similar goals and have employed a wide variety of techniques, including symbolic systems, Markov chains, evolutionary processes, and grammars [3]. In recent years, success has been reported with a variety of deep learning techniques, particularly with recurrent neural networks (RNNs) given their ability to handle data with temporal, recurrent structure, something that is highly present in music [4]. This research has proceeded to the point where there are now many high profile projects related to the field, and the technology has found it's way into the public sphere, like with Google's Bach Doodle [5].

1.2.2 Interactive Musical Agents

Across the literature, interactive musical agents are typically characterized by an input system that processes either symbolic or physical representation of the source music, an agent that takes that input and generates musically salient parameters in real-time, and an output system that takes the agent's response to play notes or control some parameter of the audio. Similarly to the broader research efforts in generative music, these systems have employed a wide variety of approaches. Further-

more in the case of interactive musical agents, these systems are often idiosyncratic in their adaptations to a particular researcher or performer.

One hallmark example of this highly-customized approach is George Lewis's *Voyager* system. Lewis is a trombonist, and he designed Voyager to track his improvisation through pitch-to-MIDI conversion and generate responses [6]. Voyager is an exceedingly intricate, state-based system that has been expanded year after year to create unanticipated musical content for the purpose of expanding Lewis's improvisational vocabulary. It has been designed solely with his performance style in mind, and therefore closely reflects the attitude and approach of its creator. Lewis's intent was to create a non-hierarchical system, and the various statistics and input fields utilized by Voyager in its abstract mappings are a direct result of this intent.

As with studies on generative music in general, some improvisational music companions have built on the recent successes of deep learning and again have found recurrent neural networks of particular use. In his work for Google Brain, Castro built an structured improvisational system that relies on recurrent neural networks for melody generation [7]. After pre-training an LSTM model, Castro utilizes the trained network in a highly-structured system that allows for the layering of drums, bass, and chords before finally entering into a call-and-response type improvisation where the user plays notes that are stored in a buffer and then inputted into the network as a primer melody to generate a "response". The system utilizes this response in a hybrid manner, as the network is used only for melodic content and its rhythmic output is discarded. Instead, the notes are sounded when the user presses a key, meaning that the timing is determined by the performer.

1.2.2.1 Improvisational Music Companions

One research goal in the field of interactive musical agents that can be found in the literature dating back to at least the early 2000s is the construction of a true improvisational music companion (IMC) that can adapt to the style of a particular

performer. In her paper on the *Band-Out-of-a-Box* system, Thom describes this goal as trying to build a system that "plays music with you, trades licks and riffs with you, improvises with you [and] ... gets to know you and your musical personality" [8]. Band-Out-of-a-Box attempts to achieve this ideal through the use of unsupervised machine learning algorithms that learns to model a musician's behavior by separating their improvisations into user-specific playing modes and then detecting playing modes and generating representative content in real-time scenarios.

The goal of IMCs represents a shortcoming of modern deep learning approaches to interactive musical agents, as the large corpus required for the proper training of a deep model means that the musical examples must be mined from large, multi-artist, multi-genre data sets. Castro argues to the contrary in his work, viewing the primer melodies as a reflection of individual style, but given that the melody network is trained on thousands of MIDI examples and uses what it learns during training to transform the primer melodies into a response, we contend that the system still falls short of the goal of IMCs [7].

1.2.3 Reinforcement Learning

The core goal of reinforcement learning research is the design of algorithms that train an agent to maximize a reward that it receives for correct behavior in an environment [9]. RL is oriented around Markov decision processes (MDPs), Figure 1.1, in which an agent performs an action based on the state of an environment and then observes a new state and, potentially, a reward in a continual loop.

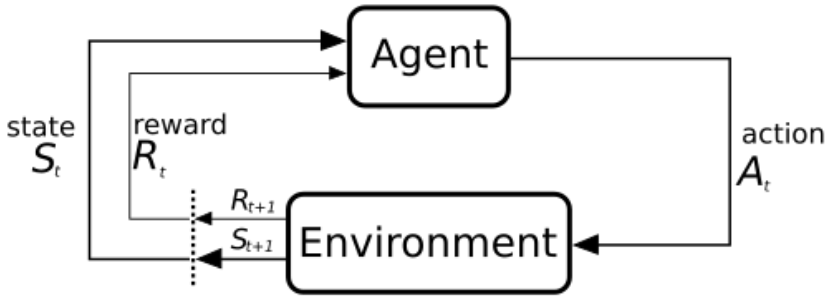

Figure 1.1: A pictorial example of a Markov decision process.

A complete Markov decision process is specified by a set of states S, a set of actions A, a transition function P that specifies a distribution over next states given a state-action pair, a reward function R, a starting state s_0, a discount factor γ that specifies the relative importance of future rewards, and a horizon H that specifies the maximum length of an episode. The goal of reinforcement learning is then defined as finding the optimal control policy in the MDP. The methods used to solve or approximate this control problem vary, and include policy iteration, the iteration of value and Q functions that specify expected reward given a policy, policy gradients, and model-free methods.

1.2.3.1 Reinforcement Learning for Generative Music

The use of reinforcement learning for generative music has been limited, but it is not entirely unexplored. Jaques iterated on previous successes in using RNNs for the generation of melodies by tuning the output of network with a RL reward function [4]. The reward function specifies certain composition rules, like avoiding excessively repeated notes, and is combined with the original RNN network to maintain the information about transitions learned by the network while also adhering to the music theoretic constraints.

In another example of the limited forays into RL for music, Collins proposes training musical agents using reinforcement learning based on rewards related to either the

predictive power of the agent or the degree of influence over the musical state exerted by the agent's actions [10]. One of the problems raised in this work, and that seems to be present in much of the research at the intersection of RL and music, is the difficulty in finding a proper reward specification for musical behavior. Encoding all of the knowledge that a musician possesses about music into a mathematically-specified reward is highly challenging, and perhaps impossible, and we posit that this challenge is one of the core factors leading to the lack of research in this area.

1.2.3.2 Inverse Reinforcement Learning

Fortunately, there has been much attention given in the reinforcement learning community in recent years on strategies for training agents in the absence of a reward function. These algorithms are collectively termed inverse reinforcement learning. Instead of training an agent to act optimally in an environment given a reward function, inverse RL recovers a reward function based on expert demonstrations in the environment, or in certain cases, optimizes a policy directly based on those demonstrations [11]. One challenge for early work in inverse reinforcement learning was the existence of degenerate solutions in the space of reward functions for which the expert trajectories are optimal. The canonical solution to this problem is the maximum entropy inverse RL algorithm, which employs the principle of maximum entropy to select among the candidate functions [12].

Since the publication of the MaxEnt RL algorithm, a variety of inverse RL approaches have found success on many simulated and real-world problems. These algorithms vary in the portions of the MDP problem that must be specified, but generally have the advantage of being highly sample efficient when compared to classical RL approaches. For example, some inverse RL algorithms require access to the underlying transition model for an MDP, while for others it is not needed.

1.2.4 Generative Adversarial Imitation Learning

One algorithm that appears to hold great promise for applications to musical domains given its ability to handle unknown dynamics, something that would be hard to specify for musical settings, and high sample efficiency is Generative Adversarial Imitation Learning (GAIL) [13]. GAIL recovers a policy directly from the expert data by finding an approximate solution to a cost-regularized version of MaxEnt RL, which induces a policy that approaches the state-action occupancy measures of the expert data. Their particular choice of cost regularizer minimizes the Jensen-Shannon divergence of the occupancy measures. This measure of divergence utilizes a discriminator function to calculate its value, and so GAIL actually relies on two networks over the course of training. Given expert trajectories τ_E sampled from an expert policy π_E, a discriminator network D with parameters ω_0, and a policy network with parameters θ_0, GAIL proceeds as follows:

- **for** $i = 0, 1, 2...$ **do:**
- Sample trajectories τ_i from π_{θ_i}
- Update ω_i to ω_{i+1} using gradient:

$$\mathbb{E}_{\tau_i}[\Delta_\omega log(D_\omega(s,a))] + \mathbb{E}_{\tau_E}[\Delta_\omega log(1 - D_\omega(s,a))]$$

- Take a policy step from θ_i to θ_{i+1}, using the trust region policy optimization rule with cost function $log(D_{w_{i+1}}(s,a))$. This is a KL-constrained natural gradient step:

$$\mathbb{E}_{\tau_i}[\Delta_\theta log\pi_\theta(s,a)Q(s,a)] - \lambda\Delta_\theta H(\pi_\theta)$$

where

$$Q(\bar{s},\bar{a}) = \mathbb{E}_{\tau_i}[log(D_{\omega+1}(s,a))|s_0 = \bar{s}, a_0 = \bar{a}]$$

We refer you to their paper for the full derivation of the algorithm and the proofs of its optimality.

1.2.4.1 Connection to GANs

This conception of inverse reinforcement learning draws parallels to Generative Adversarial Networks (GANs), which train a generator network to generate data that matches the distribution of its training set by attempting to fool a discriminator network that tries to distinguish between the two data sets. In the case of GAIL, the occupancy measure of the policy network is comparable to the data generated by the generator network, and the occupancy measure of the expert is akin to the true data distribution.

1.3 Expected Contributions

In this rest of this paper, we present *Melodic Imitator*, a generative music system designed around the generative adversarial imitation learning algorithm. We analyze the results of several experiments on potential design choices, including varying the representation of our observation space and the amount of musical information available to the agent. We also describe an interactive musical agent built with the *Melodic Imitator* system to exemplify its potential for use in live performance environments.

CHAPTER 2: MELODIC IMITATOR

The remainder of this work is devoted to the development of a new system called *Melodic Imitator*, which utilizes the GAIL algorithm to train an agent to generate melodies in the style of a particular performer. Given the sample efficiency of the GAIL algorithm, we can train an agent to operate in a high-dimensional environment with as few as 10 examples, meaning that we can personalize an agent to one individual's melodic approach as the size of the data set needed is manageable in comparison to an individual's musical output. We describe the construction of the agent, the data used, and the OpenAI gym environment we designed for this purpose. Additionally, we detail several experiments on varying the amount of musical information available in our data representation and analyze the results. Finally, we also include discussion of a live demo application which brings the project into the world of interactive musical agents and, hopefully, improvisational music companions.

2.1 Agent Construction

Our GAIL training setup includes two models: our policy network and the discriminator. The GAIL algorithm cannot properly train recurrent neural network structures, so both the policy and discriminator networks are multi-layer perceptrons (MLPs). The policy network takes in 128 units in its input layer, passes it through two hidden layers each of size 64, and then performs a softmax function to select one of 89 possible output units, one per action. Our discriminator network takes in transitions of size 217 (the concatenation of an observation and an action), passes it through one hidden layer of size 100, and outputs a binary classification. We train these networks for a total of one million steps, with a discriminator step size of 0.0003

(the policy network step size is determined in the TRPO calculation).

2.2 Data

To avoid the challenges associated with processing raw audio data, we built our agent around the symbolic musical representation provided by the MIDI standard. We composed a set of five rhythmic accompaniments and ten melodies in the key of C in the digital audio workstation Ableton Live, depicted in Figure 2.1. We chose 16 bars as the length of each of these pieces, given its common recurrence in the lengths of song sections in popular music.

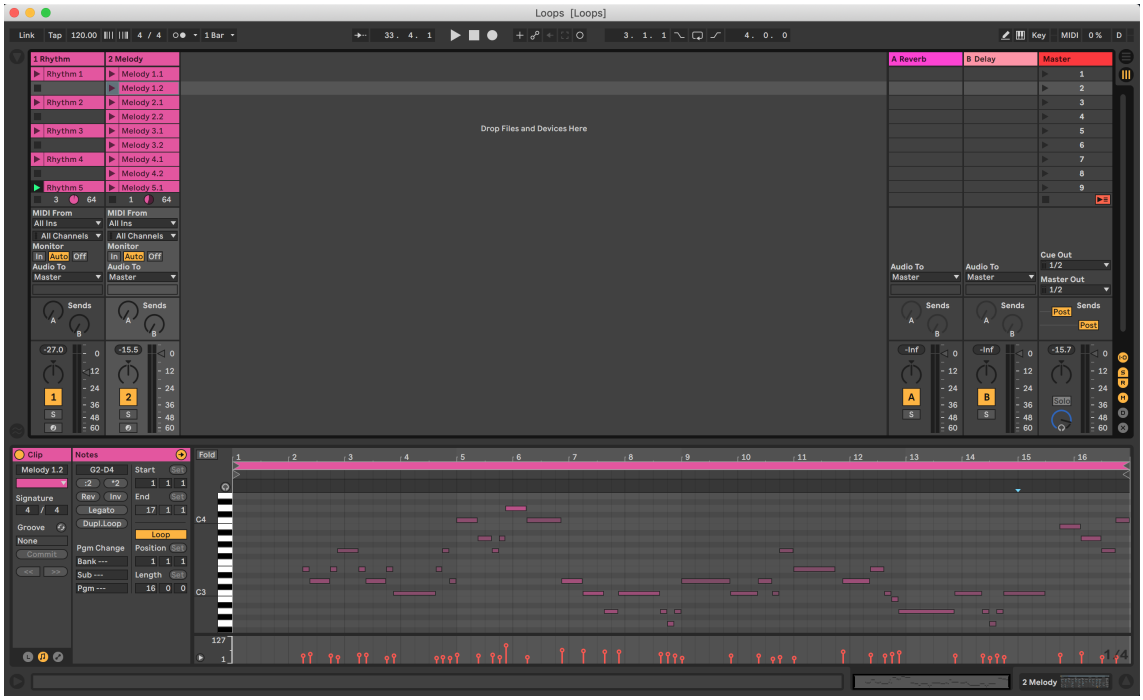

Figure 2.1: The project file for the data set composed in Ableton Live. The rhythm and melody tracks are visible in the top left, with five rhythm parts and ten melodies. One melody is highlighted in the bottom section to illustrate the symbolic representation.

2.3 Environment

To increase the portability of our work and compatibility with existing reinforcement learning code repositories, we designed an OpenAI gym specification for our musical environment. Improv_Env follows the gym framework, which requires the

implementation of a reset and step function that start and progress an episode, respectively, and the definition of an observation and action space. Our reset function loads a new rhythm example from our data set, and the step function simply returns the next 16th note frame, since the melody note action does not actually transition the environment. Our observation space consists of a vector of 128 values corresponding to each possible MIDI note, and our action space consists of 89 discrete options, one for each key on the piano plus one option for playing no note. We also implemented a render function that provides for easy parsing of the notes selected by our trained agent.

2.4 Experiments

Here we present the experiments we performed with varying the representation of our rhythmic data and adding additional structural information with the inclusion of a drum part.

2.4.1 Data Representation

We processed the raw MIDI files composed in Ableton Live using the Mido library for Python, and discretized at the precision of 16th notes. Given that each musical piece is 16 bars, we have a total of 256 time steps per episode. Each time step is represented by a vector of length 128, with one entry per MIDI note. Given that GAIL is not compatible with recurrent neural networks, we felt it was necessary to explore different methods for encoding our data to investigate the potential for increasing the musicality of our agent since we could not rely on our network structure to capture temporal relationships in the data. In total, we explored three different methods for encoding these parts. For each potential representation we also manipulated the frames to represent transposition into the keys of G and D, giving a total of 30 trajectories per experiment.

2.4.1.1 Frame-by-Frame

For our baseline encoding, we simply initialized our MIDI note vector to all zeros and changed the value to one for each note present at the given 16th note frame. This has the effect of providing our policy with observations of only what is occurring at the given frame, with no additional temporal or structural information. A visualization of this encoding process is given in Figure 2.2.

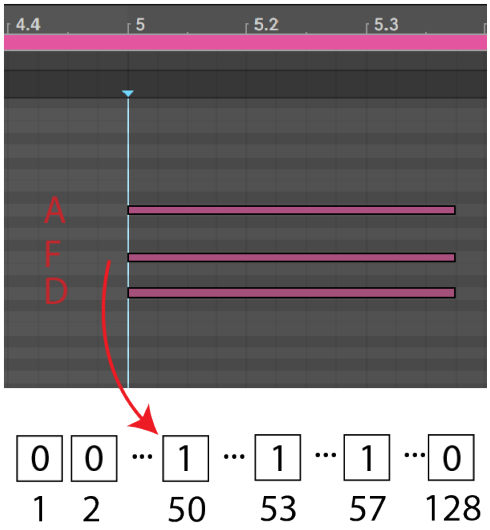

Figure 2.2: In this example, we demonstrate the conversion of our MIDI files to our baseline vector representation, with the notes sounding at the marked frame represented by ones in the vector and all other notes set to zero.

2.4.1.2 Accumulated Notes

Given that music is a highly temporal domain, with the notes previously played providing strong clues about upcoming changes or recurrences in the underlying harmonic content, we felt that it was necessary to explore potential representations that could capture this temporal information. Our first attempt at this was to simply increment the value for notes present at each frame to represent the accumulation of notes over the 16 bars. We then divided the values in each trajectory by the value of the note seen most often to normalize our observations over the range [0, 1].

2.4.1.3 Accumulated Notes with Decay

Representing the accumulation of notes over the frame provides the agent with access to temporal information about the notes played up to a given frame, but it does not provide structural information about when those notes occurred. For example, a rhythm part that plays a G chord for 4 beats then a C chord for 4 beats will have the same representation under our last scheme as one in which the order of those chords is reversed. To address this issue, we extended our previous representation by decaying the values of previous frames by a factor of $\frac{255}{256}$ at each time step. So, if a G chord is currently sounding after previously hearing a C and D chord, the constituent notes of G will be at value 1, with the notes for the C chord slightly decayed and the notes of the D chord decayed even more so. With this representation, we hoped to address what we perceived as the main shortcomings of using an MLP-based policy over an RNN-based one.

2.4.2 Additional Musical Information

After exploring the potential of increasing the musicality of our agent with additional information about the harmonic progression of an episode, we wanted to investigate possible gains from the inclusion of additional instrumental parts, in this case, a drum rhythm. Preliminary analysis on our data representation experiments indicated two things that were important for this experiment: (1) the agent had trouble generalizing across key signatures and (2) our decayed representation did not have a distribution that was easy to descend during training. To alleviate these concerns, we limited our data set to only the original trajectories in the key of C and utilized our accumulated representation as the basis on which to add our additional parts. For this experiment, we expanded our observation vectors to length 131 to represent 3 more MIDI notes for each of a bass drum, snare, and hi-hat. For each trajectory, we utilized a simple drum pattern of 8th note hi-hats, a bass drum on the 1st and

3rd beats, and a snare on the 2nd and 4th. This beat is depicted in Figure 2.3.

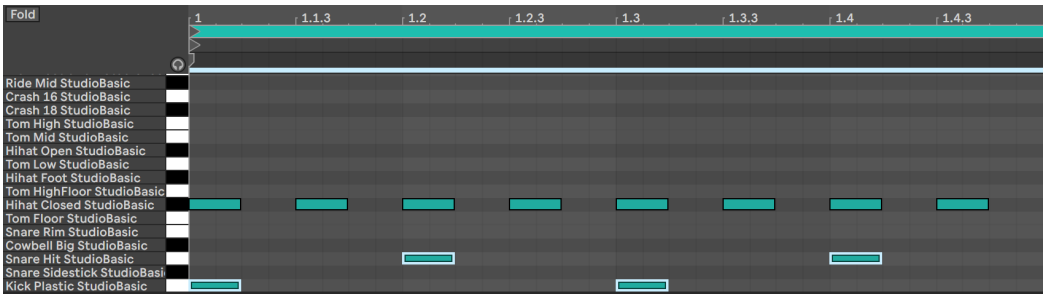

Figure 2.3: One measure of the drum pattern used for our final experiment.

2.5 Results

Given that generator and discriminator networks do not have easily interpretable and objective loss functions that can be used to evaluate their efficacy, we instead provide examples of the types of melodies generated in each experiment. Figure 2.4 depicts one of our composed rhythm parts and a sample melody for each data representation experiment. Figure 2.5 depicts the same rhythm part with a melody composed by the agent trained with the addition of drum beat information.

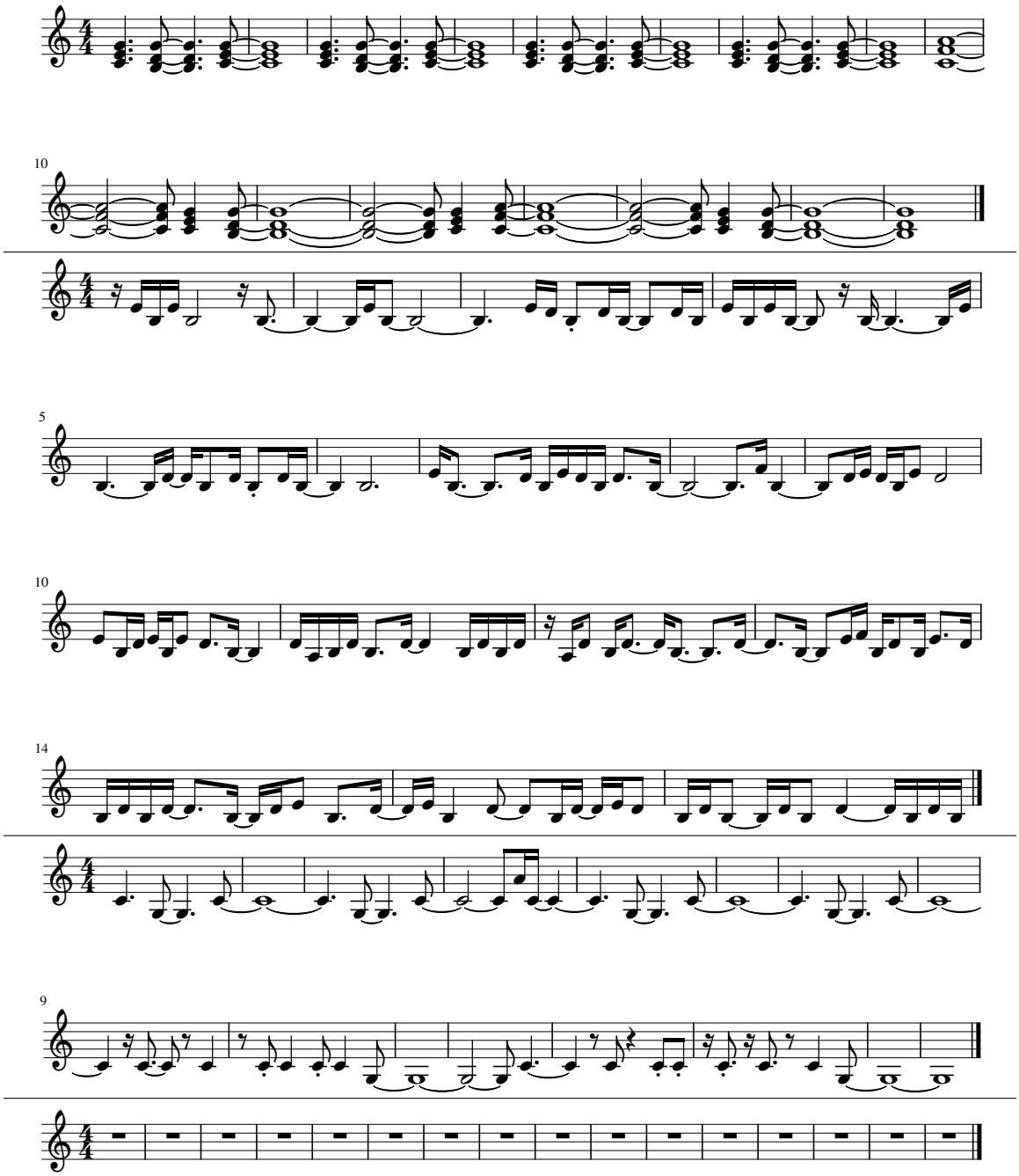

Figure 2.4: One of our composed rhythm parts and one sample melody for each of our data representation experiments in the order described above.

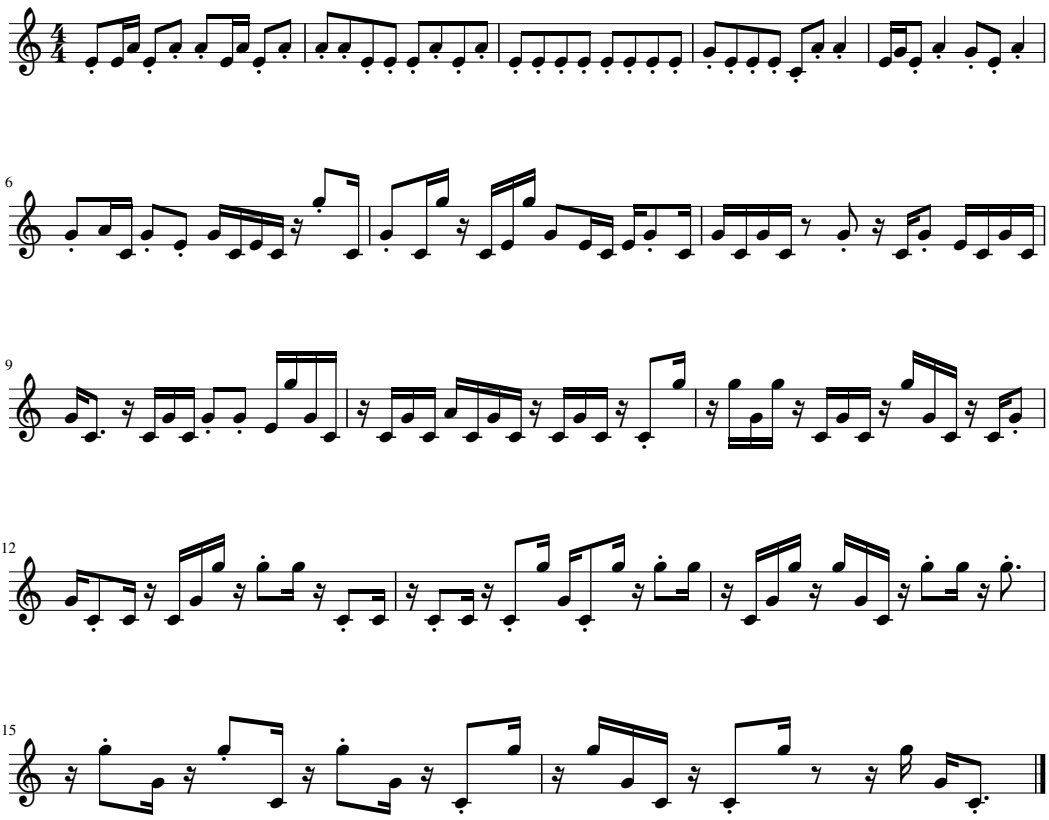

Figure 2.5: The same rhythm part and one sample melody from the agent trained with the addition of drum information.

2.6 Discussion

While none of our approaches resulted in a generative model capable of producing melodies of the same musical richness and variety as the expert examples, these results do provide several interesting avenues for analysis. Since the generative models from our first two experiments provided the most musical outputs, we will begin our discussion there.

The melodies from our baseline experiment seem to follow directly from the shortcomings we identified in that choice of representation. More specifically, given that the data representation does not provide any indication of where a particular frame falls in the overall musical structure, notes occur more frequently and haphazardly across the metre. The temporal information provided by the accumulated extension

certainly appears to have helped it generate melodies that follow a stricter sense of musical knowledge: notes occur less frequently and better follow the emphases of the underlying rhythm. Additionally, the model trained on the accumulated representation demonstrates a stronger understanding of expert data set. The motif in the first measure of the example is directly taken from the expert data set, although it is distorted slightly in time.

Interestingly, both models seemed to struggle to generalize across key signatures, and instead rely mostly on notes that are in common between the three keys. This problem of excessively repeated tokens is actually a well-documented shortcoming of generative models built with recurrent neural networks, though the extreme sample efficiency of our method in comparison with RNN-based approaches is very promising [4].

Given the results of our experiment with a decayed representation, it appears that the underlying distribution was too topologically complex for the generator to properly descend the gradient, instead settling into a local minimum where the best action choice was always to choose no note. Perhaps a representational scheme that preserves note history upon recurrence instead of resetting the decayed value to one would provide better results.

In our drums experiment, it does appear that limiting our trajectories to the key of C helped to expand the range of actions the agent is willing to take, and the drum pattern does help to anchor the action selections to mostly occur on the emphasized 8th notes. However, the highly recurrent nature of the drum pattern seems to undo some of the advantages of the accumulated representation, as notes are much more frequent than when given the rhythm pattern alone.

2.7 Live Demo System

Given our prevailing goal of using this technology in the context of a live, interactive musical agent, we designed a demo system that utilizes our model in a real-time

environment. Since we observed difficulties in the agent's ability to generate rhythmic structure, we designed a hybrid system in the vein of Castro's ML-Jam system [7]. Here we describe the interaction flow of the system.

Sessions take place within the Ableton Live environment, with an iPython notebook server running in the background. A user starts by recording an 8 bar rhythm pattern in Ableton, which is simultaneously recording the MIDI to control an internal instrument and sending that MIDI over the internal MIDI drivers to the iPython server. This server processes the incoming messages into the accumulated data representation using a Python wrapper for the RtMidi library. Once the 8 bar rhythm part has finished recording, we use our pretrained *Melodic Imitator* to provide note actions for each data frame. The user then enters into the improvisational phase of the interaction. In this phase, Ableton no longer directly records the keyboard input. Instead, RtMIDI continues listening for any note on and note off messages and uses the Mido library to route MIDI messages replaced with the note values decided on by *Melodic Imitator* back to Ableton on a different MIDI channel to control a different instrument. This two phase process is represented in Figure 2.6

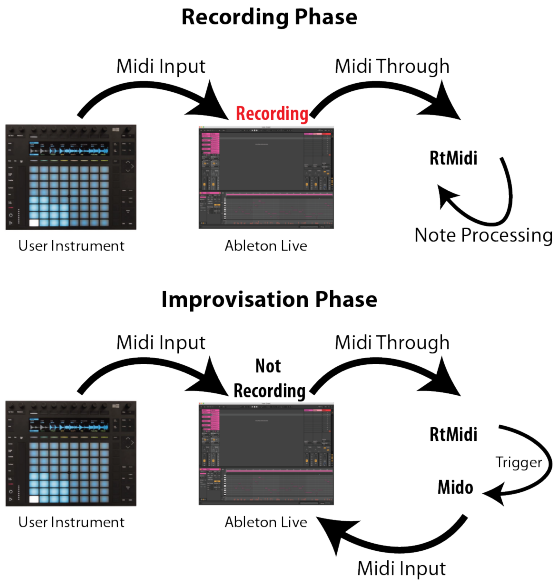

Figure 2.6: The two phase interaction flow for the live demo.

CHAPTER 3: CONCLUSION

In this section, we summarize the main conclusions of our work and look forward to future possible developments in this area.

3.1 Summary of Main Results

In this paper, we presented *Melodic Imitator*, a generative music system designed around the generative adversarial imitation learning algorithm. We explored the impacts of various data representations and of providing additional musical information on the quality of melodies generated. These results are encouraging for the potential of further explorations on the applicability of inverse reinforcement learning to musical domains. We also described an interactive musical agent built with our models, showing the possibility of the use of these approaches in real-time environments.

3.2 Limitations

While the results from *Melodic Imitator* are promising in some regards, and we were able to achieve our goal of building a interactive musical agent based solely on the style of a particular performer, there is clearly still work to do before this approach would result in a system that musicians would find useful in their creative processes. *Melodic Imitator* is currently limited in its ability to achieve high levels of musical variety, especially in terms of its potential to perform across differing key signatures. The system is also restrained in its flexibility to adapt across musical environments; in it's current iteration, it can only operate in the context of a simple harmonic accompaniment with an accompanying drum rhythm. Finally, our hybrid approach does not quite achieve the level of a true improvisational music companion given that it still necessitates performer intervention to provide the rhythmic content

of the generated melody.

3.3 Future Work

Perhaps the most promising avenue for future work would be integrating the GAIL algorithm with RNN policy networks for melody generation. Additionally, data representation is clearly incredibly consequential in the design of this type of system, and we believe that we have not yet found the ideal representation that best captures available musical information. Finally, in terms of system design, an agent such as *Melodic Imitator* would be most useful when it is maximally adaptable to any given musical environment. For example, combining the agent with audio processing methods could extend its use beyond MIDI, or enabling the agent to adapt to additional musical elements on the fly could best utilize all available information to generate better results.

REFERENCES

[1] L. Hiller and L. Isaacson, "Iliad suite score," 1957.

[2] G. A. Wiggins, P. Tyack, C. Scharff, and M. Rohrmeier, "The evolutionary roots of creativity: mechanisms and motivations," *Philosophical Transactions of the Royal Society B*, vol. 370, 2015.

[3] J. D. Fernandez and F. Vico, "Ai methods in algorithmic composition: A comprehensive survey," *Journal of Artificial Intelligence Research*, vol. 48, pp. 513–582, 2013.

[4] N. Jaques, S. Gu, R. E. Turner, and D. Eck, "Generating music by fine-tuning recurrent neural networks with reinforcement learning," in *Deep Reinforcement Learning Workshop, NIPS*, 2016.

[5] C.-Z. A. Huang, T. Cooijmans, A. Roberts, A. Courville, and D. Eck, "Counterpoint by convolution," in *International Society for Music Information Retrieval (ISMIR)*, 2017.

[6] N. Collins and J. d'Escrivan, *The Cambridge Companion to Electronic Music*. Cambridge University Press, 2007.

[7] P. S. Castro, "Performing structured improvisations with pre-trained deep learning models," *CoRR*, vol. abs/1904.13285, 2019.

[8] B. Thom, "Interactive improvisational music companionship: A user-modeling approach," *User Modeling and User-Adapted Interaction*, vol. 13, pp. 133–177, Feb. 2003.

[9] R. Sutton, *Reinforcement Learning*. The Springer International Series in Engineering and Computer Science, Springer US, 1992.

[10] N. Collins, "Reinforcement learning for live musical agents," in *ICMC*, 2008.

[11] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, (San Francisco, CA, USA), pp. 663–670, Morgan Kaufmann Publishers Inc., 2000.

[12] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proc. AAAI*, pp. 1433–1438, 2008.

[13] J. Ho and S. Ermon, "Generative adversarial imitation learning," *CoRR*, vol. abs/1606.03476, 2016.