Teaching Music Harmonization With Reinforcement Learning

Thomas Klimek & Ben Machlin

Fall 2018 Final

Abstract

This paper discusses using reinforcement learning to teach an agent to accompany a musician or a melody with harmonies. By phrasing the act of musical accompaniment as a reinforcement learning problem using states, actions, and rewards, we are able to teach an agent a policy for musical accompaniment that is applicable to any melody regardless of what it is trained on. Since there is no natural reward signal or state-space for music, this paper also details different methods of imposing these concepts and extracting different useful features from a piece of music that can be used for learning. We also propose a framework for adding a human based reinforcement signal so that our agent can learn a unique, personalized policy that is specific to the musician training the agent.

1 Introduction

Throughout history there have been two major approaches to the idea of writing and playing music. There is a rigid and structured approach used by classical musicians who memorize complex pieces of music and play with large ensembles often with impeccable precision and attention to timing and detail. There is also a looser and more free-flowing approach utilized by jazz musicians who utilize their immense knowledge of musical theory and apply it through improvisation in an act of listening and responding to their band, audience, and general environment. While programming a computer to take the first approach of classical musicians can simply be boiled down to encoding a set of instructions and timings, the idea of teaching a computer to improvise is much more nuanced and interesting.

The tool utilized for problems like learning musical improvisation, where the learner must consider its environment and interactions in a real time setting, is reinforcement learning. Reinforcement learning is a machine learning paradigm which seeks to train an interactive, goal seeking agent a policy to maximize reward signal in an unknown environment. By modeling our environment as a partially observable Markov decision process, and our problem as an optimal control problem, we can take a computational approach to solving this problem for tasks like musical improvisation and accompaniment.

This paper details the use of reinforcement learning techniques applied to music, as well as exploring different formulations of music as a reinforcement learning problem. Since the concept of what makes sound music, and more importantly what makes "good" music, is highly subjective, there can be no implicit formulation of state and reward that is agreed upon by every musician, but rather this formulation like music is highly subjective itself. We propose one interesting way of framing music as a reinforcement learning problem using function approximation and feature extraction in conjunction with both theory based and human based reward signals to create a learning model. Due to the aforementioned subjectivity, the model is not a complete model for the theory of music, however a model for the theory of a specific *musician* through our framework. We tackle the problem of learning from a predefined and static piece of music, like a classical musician, and extrapolate this policy to be utilized in improvisation, like a jazz musician.

2 Related Works

2.1 Music Plus One and Machine Learning

One paper related to our music accompaniment system is Music Plus One and Machine Learning [3]. In this paper, researchers describe a machine learning accompaniment system that similarly learns from a soloist. This system is composed of three modules; the first computes a real-time score by listening, the second generates the output audio by applying transformations to the input, and the third connects these two modules by predicting future timings. We follow a similar framework where the three key processes behind our system consist of listening to the

input and feature extraction, generating output, and playback which links the input and output. They describe these subtasks in this paper as “listen”, “Predict”, and “Play”. Our system is also based around these subtasks however varies in the method of audio processing.

The Music Plus One system functions off raw audio files, performing varied frequency analysis to extract useful features for learning, however we chose to work with MIDI files. MIDI is a standard encoding for electronic music which carries information that specify note pitch, offset (timing), and more. By working with MIDI our system does not need to perform frequency analysis, but rather can more directly apply rules of music theory and learn relationships between known note values. This also imposes a limitation to our system in that it requires MIDI signal to perform learning. While this makes learning from a MIDI file much easier and more efficient, it will not function with audio files or live instruments that can not be encoded into MIDI format. This is discussed in more detail in the Conclusion and Future Works section.

2.2 Reinforcement Learning for Live Musical Agents

Another paper in the field of music based learning that we have used as a resource is Reinforcement Learning for Live Musical Agents [4]. This paper details a system called *Improvagent*, which similarly seeks to make musical predictions by modelling musical input as a dynamic state-action case library for MIDI based improvisation. Essentially the *Improvagent* system creates a current state for a piece of music based on a melody being played, and considers the next note to play as an action generating reward. This paper uses a few methods of reward signal, one of which being a reward signal generated by human musicians, in which they predict the next note and the prediction of the model is then compared against this. We found this to be a crucial concept for accounting for subjectivity in music, and thus utilized a TAMER framework to implement human reward.

This paper also suggests useful ideas for a Sarsa based learning algorithm that utilizes feature extraction. By modeling the current state of a song based on features of what has already been played, we end up with a very large state space with sparse data for learning. This problem is addressed through feature extraction and using a variation of K-nearest-neighbors algorithm to generate a series of predictions for the next state. Some of their features consist of counts of different pitches being played as well as scale analysis to try to match the melody to a musical mode such as major, minor, or chromatic. We expand upon feature extraction in our system creating more detailed features to represent what has been played in terms of musical theory, however we really on function approximation using a linear update rather than the KNN approach to assign Q-values to unvisited states.

2.3 Tamer

Lastly, we utilize the TAMER framework detailed in the paper Interactively Shaping Agents via Human Reinforcement [2]. This paper describes a framework for shaping an agent’s policy via evaluative human reinforcement signal. Thus the goal is to maximize the prediction

for the human reinforcement signal. While our algorithm follows more along the lines of the Sarsa detailed in Sutton and Barto, we utilize human reinforcement signal to create personalized policies. In this sense our agent can be trained to a specific musician based on their given reinforcement signal. This paper reports that human reinforced learners generate significantly more reward in a much shorter period of time, however cease learning after the human training is done. This conveys the major limitation of the TAMER framework, that it is entirely reliant on human training and will eventually be outperformed by traditional reinforcement learning algorithms. For this reason we chose to incorporate both human reward and an observed environmental reward signal so our agent can continue to learner after the training has been completed.

3 Technical Approach

3.1 State Action and Reward

The key concept behind any reinforcement learning approach is to be able to phrase your problem in terms of states, actions, and rewards. While some domains have an implicit notion of state,action or reward, the domain of music improvisation requires more creativity. It is important to note that there are very many ways to define these concepts for a musical piece, and no one formulation is more correct than the other, however some might be more efficient for reinforcement learning. That being said, our formulation is informed by our research and personal knowledge of music theory and comes with its respective strengths and weaknesses.

To rigidly define these concepts we will start with the state space. Our model of the environment represents a state as a series of notes that have been played in the melody in one measure of music. For simplicity we consider music with 4 beats in a measure, and up to 8 notes per beat. We also only consider the true value of the pitch, meaning the octave of the note is ignored, for a total of up to 32 notes per measure each being one of 12 different pitches. This means the upper bound of our state space is 12^{32} = 3.42E34.

Because our goal is to create harmonization, our actions will consist of playing chord. Because there are technically very many chords when considering concepts like voicings, inversions, and extensions, so we simplify our actions to be chords consisting of three notes. Because there exists a chord centered at each pitch in the chromatic scale, the total amount of possible actions are 12 major chords + 12 minor chords + 12 diminished chords + 12 augmented chords = 48 total actions. Adding modularity to our chord representation (e.g. incorporating complex chords of more than 3 notes, or inversions) would possibly lead to a more thorough model. However attempts to achieve this led to complications that compromised the amount of attention we could put into other aspects. As will be discussed later, this is likely a fruitful area for development.

For generating reward we introduced two methods to the environment one based on correctly knowing the chords of a song, and hence the actions to take, and the other based on the

TAMER framework. Due to the fact we also wanted to do a large amount of non-TAMER training for our agent, we utilized training over the correct chords for various songs by splitting the MIDI files into a melody and harmony track, then using the melody to construct the states and features, while using the harmony to compare against the selected action for reward. The overall architecture of our system can be best understood by looking at the graphic below.

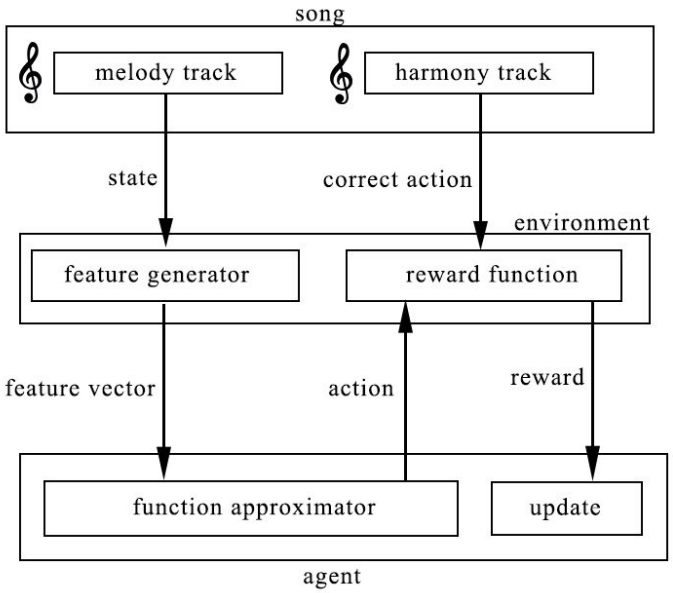

Figure 1: Program Diagram

Figure 1 describes the flow of our program through the different modules we have implemented. It is worth noting the the actions do not influence the state transitions which are implicit to the melody track. Rather we have a defined flow of states, and our actions generate different reward for each state. The reward function in the chart compares the selected action of the function approximator to the correct action from the harmony track and returns either a 0 or 1 reward for an incorrect or correct prediction, respectively. Utilizing a TAMER framework simply requires changing the reward function to look at the action and return a reward based on a user input.

3.2 Learning Algorithm

The learning algorithm used for our agent is best described as an on policy control algorithm using function approximation. We follow the general pattern of an episodic Sarsa for estimating $\hat{q} \approx q_*$ similar to what is described on page 198 of Sutton and Barto. We use a linear rules for the function approximation following the formula $\hat{q} = w^T x(s,a) = \sum w_i * x_i$ where *x(s,a)* is our function for generating a feature vector *x(s,a)*: (State, Action) → Feature Vector

Sarsa with function approximation

Initialize weights arbitrarily
Initialize Q values to 0
For each step in an episode
if updates exist
update weights
update Q values → using function approximator
if ε probability #explore
action = random action
else #greedy
action = max(Q_values)
reward, state = environment.take_action(action)
updates = $(\gamma * Q[action]) + reward - Q[action]$

3.3 Feature Generation

For our function approximator, we generate a set of features over each action and use these features to estimate q-values of each action. This ultimately informs which action is selected by our agent, so it is essential we use good features for our agent to learn with. We generate a variety of features based around music theory and simple mathematical comparisons, which we later experiment with for learning. The features are described in detail in the following chart.

Table 1: Feature Descriptions

count_feature(s,a)	Returns the percentage of notes within a state that are shared with a given action.
shared_feature(s,a)	Returns the percentage of notes in a given action that are shared by the state.
tritone_feature(s,a)	Returns 1 if the state does *not* have a tritone with the given action, and 0 if it does.
last_bar_feature(s,a)	Returns an average of the values of all features generated for the previous state given the action.
root_feature(s,a)	Returns 1 if the root appears in the state for a given action and 0 otherwise *(the root describes the tonal center of the chord and thus can often indicate the given chord for a melody).*

third_feature(s,a)	Returns 1 if the third appears in the state for a given action and 0 otherwise *(the third is the first harmony that informs the quality of the chord i.e. major, minor…).*
mode_feature(s,a)	Identifies the mode of the state, i.e. the most common note played, and matches it to actions. Returns 1 for actions within the same mode and 0 otherwise.
previous_same_feature(s,a)	This feature is used to determine if the same action has been selected recently. Returns 0.5 if the action was selected for the previous state, 1 if it was selected for the past 2 states, and 0 otherwise.
previous_scale_feature(s,a)	Determines if the the given action would fall in the previous action's scale or vice-versa.
previous_2_scale_feature(s,a)	Determines if the given action and the past 2 actions could all fall into a scale. Returns 1 if they share a scale, 0 if they don't.

4 Experiments and Results

4.1 Experiment 1: Feature Comparison

In attempts to keep our distance from a reliance on music theory for learning, we tested our many features. Some completely based on common sense with little music theory dependence and some using basic music theory ideas.

This experiment is exploratory in nature and its purpose is to isolate the best combination of features for fast and optimal learning as well as to make us question what factors determine 'proper' harmony. Each result of this experiment was obtained by simulation of 100 episodes, averaged over 10 trials. The training corpus consisted of a handful of midi song fragments. The entire corpus was iterated over in each episode. We tested each feature set with two songs: Starboy by The Weeknd and Despacito by Luis Fonsi. Refer to Figure 2 for reference.

To begin we test each feature in isolation. Tests 1 through 8 test most of our features on their own, beginning with no features. Because any given key contains 7 of 12 possible notes we expected no features to produce In Key results much less than 7/12 = 0.5833. Additionally, the likelihood of guessing a chord correctly is 1/48 = 0.02 and guessing a function correctly is ~ 2/48 = 0.0416. The interesting features to note in the isolation experiment are the count and shared features. These are the most basic comparison of two sets of notes and our frontrunners so far.

Combining count and share features yields promising but still pretty subpar results. Getting <20% correct chords for Starboy and <40% correct for Despacito isn't ideal. Though already, the chords being generated fall over 80% inside the scale of the song. Remember, the agent has no concept of what a scale or key is and so getting the In Key measure up is a good result.

We continue combining features and observing results to see how much we can improve on our base set of count+share features. We see that the mode feature and the root feature don't do much good for our scores and may even hurt the learner. Theory-wise, this indicates that the melody is often not just a revoicing of the chord notes. For example, a C major (C+E+G) chord's corresponding melody if often not just an assortment of Cs and Es and Gs. The appealing features in tests 10 through 15 are 'tritone' and 'last'.

The tritone feature dips a bit into music theory in that it detects if the given chord creates a tritone with any note in the current melody. A tritone is an interval of 6 half steps. This splits the octave in half and creates a lot of dissonance. The tritone used to be referred to as the devil's interval but has found some use in less consonant genres. But because our corpus is largely pop songs (i.e. have relatively simple melodic and harmonic structure), the tritone feature seems to improve our agent. The 'last' feature is interesting because it uses the previous measure's melody. It returns an average of the feature calculations for the given chord if applied to the previous measure of melody notes. This is one way of taking into account musical context without using music theory. The positive effect of this feature implies that a chord that sounds good with the last measure will likely sound good with the current measure.

				Starboy			Despacito		
Test #	Episodes	Trials	Features	Correct	Function	In Key	Correct	Function	In Key
1	100	10	none	0.015	0.0263	0.5893	0.0125	0.05	0.6083
2	100	10	count	0.10878	0.1518	0.7964	0.3937	0.3969	0.8354
3	100	10	shared	0.1575	0.2269	0.8429	0.2625	0.2906	0.8104
4	100	10	mode	0.01125	0.0263	0.5786	0.01875	0.025	0.5771
5	100	10	tritone	0.0375	0.0694	0.7286	0.0375	0.1156	0.7063
6	100	10	root	0.04125	0.0713	0.6893	0.0125	0.02187	0.6729
7	100	10	same	0.02625	0.045	0.581	0.0125	0.02813	0.5792
8	100	10	pscale	0.03	0.0525	0.5952	0.01875	0.05625	0.6062
9	100	10	count,shared	0.1238	0.1632	0.844	0.3812	0.3875	0.825
10	100	10	count,last	0.11246	0.1613	0.8429	0.6125	0.6125	0.9292
11	100	10	shared,last	0.14627	0.2269	0.8976	0.475	0.5125	0.9417
12	100	10	count,shared,last	0.14249	0.2175	0.9036	0.6125	0.6125	0.9292
13	100	10	count,shared,mode	0.0975	0.12	0.8262	0.3937	0.4062	0.8417
14	100	10	count,shared,tritone	0.12002	0.1744	0.9012	0.475	0.5437	0.8958
15	100	10	count,shared,tritone,mode	0.09375	0.1331	0.8845	0.45	0.5156	0.8875
16	100	10	count,shared,tritone,last	0.13871	0.2344	0.9607	0.6875	0.75	1
17	100	10	count,shared,tritone,root,last	0.11246	0.1463	0.9607	0.3937	0.4562	0.9167
18	100	10	count,shared,tritone,same,last	0.14249	0.2269	0.9548	0.6875	0.75	1
19	100	10	count,shared,tritone,same,last	0.13871	0.2269	0.9548	0.6875	0.75	1
20	50	10	count,shared,tritone,p2scale,last	0.14627	0.2381	0.9583	0.6875	0.75	1

Figure 2: Results of experiment 1

The pscale and p2scale features are two theory related features that look at the last selected chord and the one before it and calculate the likelihood that they are in the same scale or key. Their effect doesn't seem to hurt the performance but because of our desire to stay away

from music theory and the computational intensity of these features, we did not select them for our optimal feature set.

Using these tests, combined with the qualitative amount of time each test took, the best feature set we determined was in test 16: count, share, tritone, last. This captures the minimal amount of computation with a top tier score for both test songs. We also notice how Despacito seems to plateau and Starboy never quite gets properly learned. Perhaps we could search for more features, or perhaps we could experiment with an alternative reward signal. Experiment 2 does the latter.

4.2 Experiment 2: TAMER

In an attempt to coax our learner into better performance, or at least more personalized performance, we implement a TAMER system. As seen in the TAMER paper, this reward system has the potential to learn much faster and possibly in a more nuanced manner that with our previous reward signal.

Test #	Train Eps	TAMER Eps	Initial Weight Values	Starboy			Despacito		
1	100	3	0.25	0.1071	0.1964	0.9762	0.6875	0.75	1
2	100	3	0.01	0.1071	0.1786	0.9286	0.4375	0.4375	0.8125
3	100	3	0.0001	0.1429	0.1786	0.9257	0.5625	0.7188	0.9583
4	100	3	0.000001	0.1429	0.2321	0.9524	0.6875	0.75	1
5	100	3	0	0	0	0.2292	0	0	0.2292

Figure 3: TAMER experiment results

The first attempt of this experiment used initial weights of 0, just like in experiment 1. However, the problem arose that the random selection of chords rarely produced a passable choice and so the agent would only receive negative feedback and spiral away from learning.

So to compensate we began with our weights at 0.25. We thought that if we had chosen our features well, it wouldn't be invalid to given them a nudge in the right direction to start. Taming on these weights seemed to work pretty well. The results were no better than in experiment 1, but not much worse. The Starboy chord selections tended to be a lot of repeats, albeit in key, but very safe choices. Despacito chords settled into their plateau from experiment 1. Hoping to gain more control over the final policy, we decreased our initial weights more and more. The initial chord selection for both songs was mostly identical between tests, but with lower initial weights it did become easier to shape the policy. However, it became obvious that the shaping did not have a large effect. Both songs would settle into familiar patterns and only a few precarious chords would be affected by the shaping. With the weights initialized to 0.0000001 we had some luck with Starboy and maxed out with Despacito - a good sign for sure. However, the actually playback didn't sound all that much better. To get a baseline, we went back and completed the 0 value weight test (5). It managed to learn to be even worse than random!

This experiment yielded interesting and useful results, if only in what to improve rather than how to do it. Firstly and importantly, it indicated that our features are insufficient. Even for two simple songs with simple chord progressions, the features could not be learned or tamed into a state that could produce these progressions. Even for Despacito, when we learned more the ⅔ correct chords, it could not find a way to fix the incorrect chords without losing the correct chords. This tells us that the feature set needs to have more features so that the agent can distinguished between similar states.

Another key takeaway is that perhaps this problem is not yet ready for TAMER integration. When the agent did well, taming didn't seem to have a huge affect, and when it did poorly taming could not bring it out of the hole. It would likely take a significantly more time taming to produce desirable results. However, this is one weakness of human reinforcement - it's tedious and boring at times and in this case took a long time per trial. Perhaps with a cleaner and faster framework and a better feature set, taming would be a useful system, but until then, a traditional learner is just as good or better.

5 Conclusion and Future Work

While this project did yield interesting experiments and results, it also left us with a lot of room for future work and improvement. From a theoretical standpoint our two main areas of improvement would be creating a better representation and then experimenting with different learning algorithms. What we learned through experimentation is that because our states don't encode data from previous measures of music, it is difficult to learn a policy for song as a whole, and instead we just perform according to a single isolated measure. It also caused issues for TAMER training as our features where not appropriate to learn the type of policy we had envisioned. Thus, the main area of improvement and future work here would be to rework both the notion of state representation and feature generation. Beyond this it would be interesting to experiment with non-linear updates and different reinforcement learning algorithms for on policy control, however these experiments will not provide valuable until we can create a complete and efficient feature representation.

From an implementation standpoint, there are also a few areas for future work in order to make our system more modular and increase usability. One of these areas comes from our processing of music input and encoding it into our environment. We set some limiting restrictions while implementing this in order to simplify our results and debugging process, however in the future we would like to add more options for songs for input, more possible chord and action choices, adn a more advanced rhythm system.We used the assumption that the rhythm of the agent's output would be fixed, however we could use a seperate learning algorithm to generate predictive rhythms. We would also like to develop a separate module for real time audio processing as ours relies on preprocessing the entire file to parse it to the environment module. These implementation changes would simply add more possibilities for using our system, and could lead to more interesting experiments.

Overall this was an interesting experiment in implementing reinforcement learning techniques and performing analysis on music. Luckily for us, even when our agent made wrong predictions this still lead to interesting sounds being generated. This is primarily because the rules of music theory are much less like the laws of physics, and more like guidelines. Thus even though our system occasionally broke those rules, it still produced pleasant sounding and acceptable chords over the melody. This project also lead a lot of interesting ideas in terms of processing music, and modeling the cognitive behavior of humans for musical improvisation that could be later expanded. Despite achieving our goals of creating a working system and conducting meaningful experiments about different representations of musical data, our project can still be expanded in many interesting ways to learn more about the joint field of music and reinforcement learning.

6 References

[1] Richard S. Sutton and Andrew G. Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[2] W. Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16. ACM, 2009.

[3] Raphael, Christopher. "Music Plus One and Machine Learning." *ICML*. 2010.

[4] Collins, Nick. "Reinforcement Learning for Live Musical Agents." *ICMC*. 2008.