Matt Rubin, Brian O'Connor, Deniz Kahramaner
http://www.melodydragon.com/

Research Project Report: Melody Dragon

I. Introduction

Melody Dragon is an online tool that generates melodies based on a Q-learning algorithm, improving the quality of its music based on user feedback.

We based our research on the hypothesis that there are certain melody patterns that are universally accepted as good, such those in the famous classical compositions of Bach, Beethoven and Mozart. While not everyone is a fan of classical music, the works of the most famous composers are generally agreed to be quality music.

There have been many attempts to theoretically construct the perfect melodies using music theory and mathematical methods. While such research is theoretically sound, it does not take into account any sort of creative impulse. In a computational environment, such creativity could be replicated by randomized melody generation, constrained by certain musical patterns. We posited that the average person, while perhaps unskilled at music composition, has enough innate musical knowledge and acquired musical taste to judge the quality of a short melodic pattern. Our approach is to take user feedback on short randomly generated melody fragments to form a policy with which a computer can construct good melodies.

This is an interesting problem for a number of reasons. Practically, automated melody generation could be very useful and lucrative tool. The business of music production is a multi-million dollar industry which depends human creativity for all but the simplest and most generic of compositions. However, if an artificial agent were developed which could generate likable melodies, it could revolutionize the music industry.

The problem is also conceptually interesting. Creativity is a very difficult quality to define computationally and to imitate in an artificial agent, and we are attempting to replicate it on a small scale by offering every possibility and then crowd-sourcing the decision-making process. While a musically theoretical approach to the problem could produce a melody with greater complexity, by simplifying the problem, we can use the combined creativity of hundreds of people to form a generalized aesthetic model with a wider range of creative possibilities. Given the proper time and resources, this machine learning model could expand to more complex models of creative composition.

Due to the immense complexity of musical composition, a model capable of generating every possible combination of notes, chords, and rhythms has thousands of variables and is far beyond the scope of our project. The approach we took was to remove the complexity of rhythm and give the user a single sequence of quarter notes. We begin by presenting the user with a seed melody of 3 notes and the ability to "like" or "dislike" the melody. If the user likes the melody, a new note is added to the end and they are asked to rate the expanded composition. If at any point they dislike the direction the melody has taken, they can click the "dislike" button to rewind one step and grow the melody in a new direction. In this way, the agent will learn over time which melody patterns a user likes. Using the last 3 notes of a melody as the state and the following notes as potential action, we can generate Q-values for each pattern. In this way, user feedback trains an initially-random agent to follow a policy of user-preferred patterns.

After a large amount of user feedback, the agent will have learned enough to be consistent in creating good melodies. The Q-values may converge, but additional user feedback, randomized start states, and a transition algorithm based on weighted probabilities rather than a fixed policy ensure that the agent will not always produce the same melody, but it will consistently produce a melody that all (or a very large majority) of its users should like. *We found that a few hundred training sessions was all that is required to considerably improve the quality and general appeal of the melodies generated.*

II. Prior Work

Most of the research done in music and machine learning is to use machine learning to model specific musical styles. Then, there are a series of research papers that use these models to emulate musical styles for music generation.

In his paper, Darrell Conklin **[1]** talks about the different models researchers use to model music. We considered most of these models before we designed the AI in our project. First, Conklin's paper talks about the "Random Walk Method" to generate music. The idea of this method is to get a random event from a distribution of musical events obtained by analyzing many songs considered as successful, and add this random event to the melody at hand. This method will not work as well as our project because of two reasons:

1) The random walk method will have a hard time creating complete pieces because it concatenates random events together that might not necessarily sound good together.
2) Our algorithm does not have this problem because we get user feedback for every note transition in the melody, which will guarantee that our trained agent will consider the overall unity of the melody.

Allan **[2]** talks about using the Viterbi algorithm for music generation by decoding a set of given observed states in a Hidden Markov Model. Allan uses "harmonic symbols" as the hidden states and the given melody is used as the observable states. The main drawback of this approach is that it is computationally expensive for long melodies, which is not practical for our server-client model where multiple clients can train the server a step at a time. For such a model, the server's computation time is likely to exhaust the casual user's patience. Both Conklin's and Allan's papers argue that because of this drawback of the Viterbi algorithm in analyzing melodies, heuristic search and control strategies must be applied for better music generation.

Jiménez **[3]** finds a solution for this, by using A* search for the n best states. However, this also posed as a problem for our project, because of two reasons:

1) A system that creates a few high-probability pieces cannot be called creative. According to public opinion, there isn't one good melody; there are many. By using reinforcement learning, our algorithm eventually converges, but the constant user input would prevent it from converging to only one final melody.
2) We wanted to base our agent on user feedback, so we decided reinforcement learning with Q-values that got updated according to user feedback could produce better results.

Bruce Jacob **[4]** uses genetic algorithms to create melodies. He uses the "valid vertical pitch combinations" that range from one to twelve as chromosomes. Each allele (an element in a DNA sequence) represents a single note or chord, and adjacent alleles represent a valid transition to the left or right of a score, where. Thus, a chromosome is used to validate a phrase of music (called patterns in this paper). Accepted phrases get sent to an "ear" module that checks the tonality of the phrases and sends it to an arranger that merges the phrases and sends them to a human evaluator. This paper persuaded us to run every one of our transitions by human evaluators (users of the web page) because it will keep improving the transition functions, like it does with Jacob's agent.

III. Approach

Melody Representation

Every melody is represented by a sequence of numbers that are directly related to pitch and octave. (The numerical value equals the octave times twelve plus the individual pitch, where C is 0, C sharp is 1, D is 2, … and B is 11.) We used the MIT's open source music21 Python library (http://mit.edu/music21/) to convert a MIDI file to a stream and then our own code to map music streams to states. The final three notes of a melody form a state, where the first note is normalized to a common value, and the pitch offset of the remaining notes, as integers, define the state. We should note that while a mapping from a note to an integer is bijective, mapping from notes to states is non-injective, but it is surjective.

Artificial Intelligence

Our problem has a limited number of well-defined states and transitions; however, the reward function and policy are unknown. When deciding what would be the best way to approach to problem, we decided that Q-learning would be most effective because of its special property: given decent learning parameters and sufficient exploration, it will converge to the optimal policy without ever needing actually follow it. It would be impossible to visit every possible melody, much less to derive enough information to learn anything, so important characteristics of the melody is represented in the generalized states described above. Q-values are updated in the standard fashion incorporating the old value as well as the reward as defined by the user as well as the utility of the new state. We also keep a value for each state, as defined by users reward, used for generating the start state.

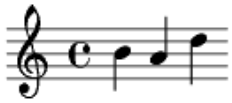
Seeding the Training Model

To train our agent in addition to the user feedback given, we decided to take famous classical pieces from Bach, Beethoven, Mozart, Vivaldi, etc. to seed our database, as explained in the introduction and in our results.

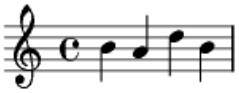
The music21 library contains many classical songs that we extracted patterns from to train our agent. The patterns are treated like the note by note melodies generated by the web page; training can be viewed as sliding a four-note window over each melody, treating the first three notes as the state and using the fourth note as defining an action. The transitions in each pattern are given positive feedback, as the entire patten is assumed to be "good".

Online Training: Example of Functionality of a Running Session

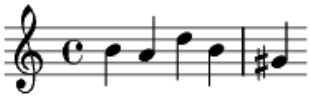
1) Start state: The start state is any valid state. At the very beginning before any training, they are generated randomly. After some user feedback, the start state (the three notes) are still chosen weighted-randomly, where the weight if each state is taken from feedback learned during the training process.



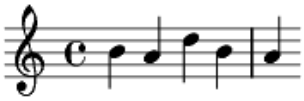
2) State 2 after the user "likes" the initial notes: After the user clicks the "like" button, positive feedback is given to the start state, and a note is appended to the melody. The note appended to the melody is provided by an action generation function that provides the action with the highest Q-value 90% of the time, and a random action the remaining 10% of the time. This ensures that we will explore all of the possible states.



3) State 3 after the user "likes" again: After the user clicks the "like" button, the Q-value for state 2 and last action is updated with a positive reward, and we call our action generation function to generate the next note.



4) State 4 after the user "dislikes". The Q-value for state 3 and the last action is updated with a negative reward. We then remove the offending note, and generate another pitch using the action generation function, taking care not to repeat the same note.



5) After the user clicks "I am done!": The current melody is considered a terminal state and the Q-value for the previous state and last action is updated with a larger reward. The user is given a unique link which can be used to access this specific melody in the future.

IV. Results

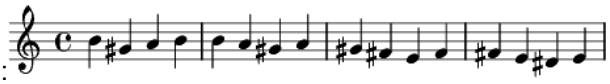
In such a creative project, there are a few different ways to measure results. Qualitatively, though the value of a particular melody is subjective, we found that the algorithm improved over time, producing more fluid consonant melodies as its training progressed. While the initial melodies generated were entirely random, over time a number of patterns appeared which the agent had learned were melodically pleasing.

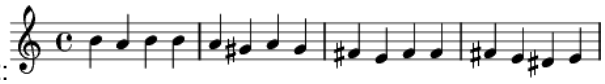
Quantitatively, we measured success in two ways: the ratio of "likes" to "dislikes" and the average

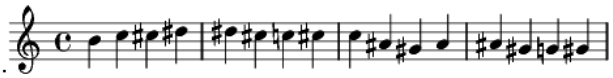
length of a user session. Improvements to the like/dislike ratio indicate that the users are happier with the melodies the program is generating. A less direct measure of success is the length of the user session; longer overall session lengths indicate that users are more engaged by and attached to the melodies that they are generating and a longer generated melody indicate that a user is pleased with the sequence of notes generated.

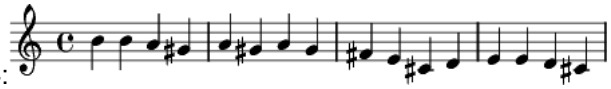
Qualitative Improvement

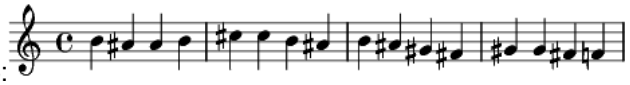
Here are some examples of generated melodies that show specific patterns found after extensive training.

1: 

2: 

3: 

4: 

5: 

Before thousands of pieces of user feedback, all the generated notes were random and the average user would easily know that the notes did not go well together. After training the agent, one can detect specific patterns that sound good in some of the melodies. These patterns are created by multiple users liking very specific transitions and thus rewarding them.

For example, melody 1 and melody 2 have a similar pattern of transitions after the first couple of notes. However, this sequence is only triggered by a very specific combination of start states (as can be seen, some patterns repeat but never repeat many times).

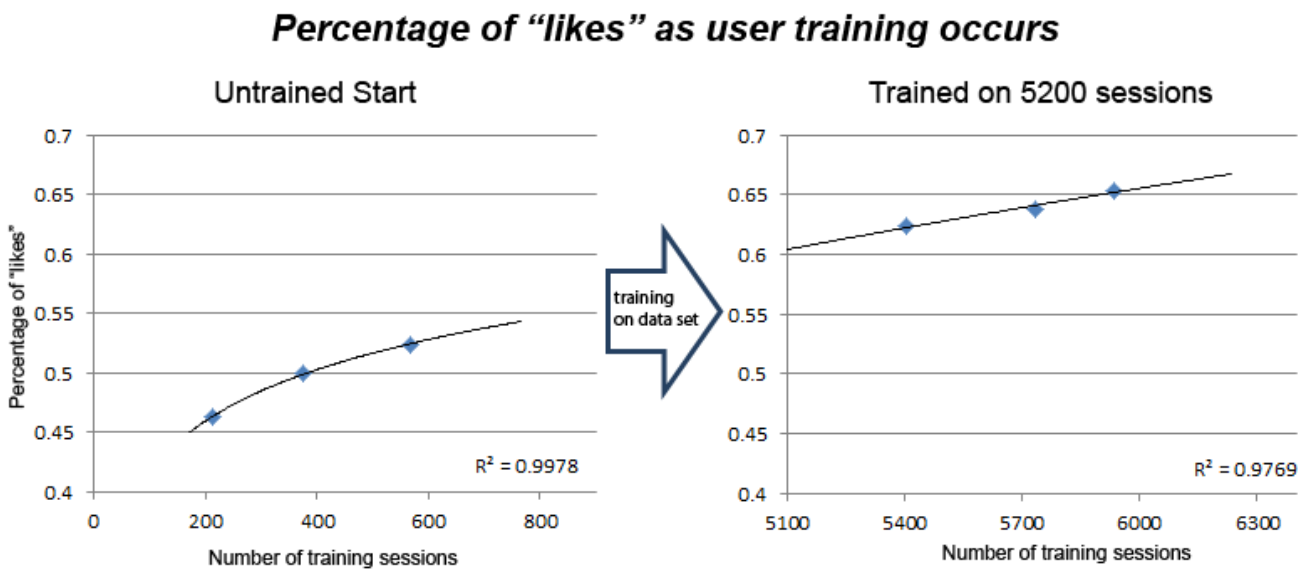
Melodies 3, 4 and 5 all have a gradual decrease in pitch towards the end of the melody. However, each of these patterns are unique from each other due to the distinct Q-values in every scenario. In other words, none of the patterns are exact copies of each other. As generated patterns are rearranged and shifted up or down on the staff, each generated pattern is inherently unique.

Quantitative Improvement

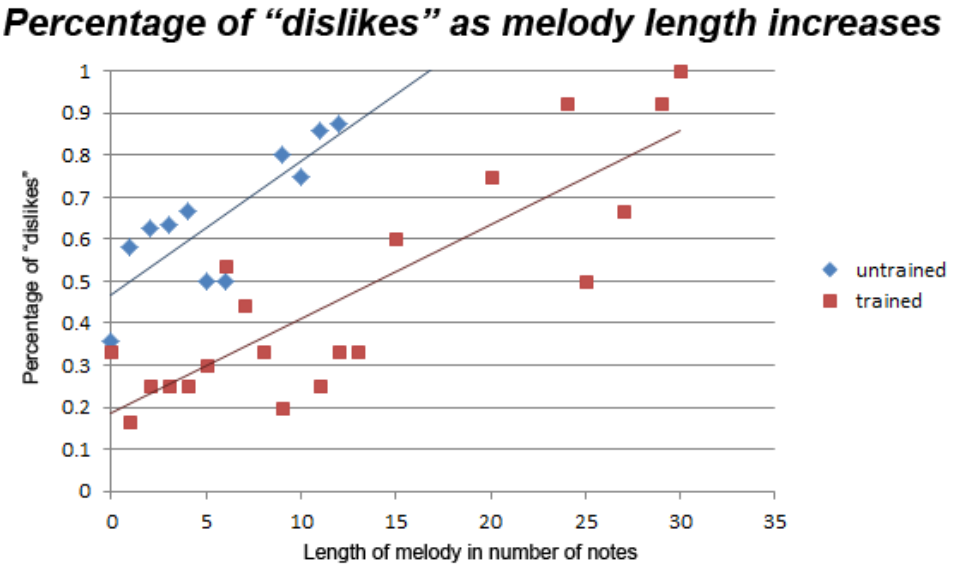
The “like” and “dislike” buttons in the user interface serve two purposes: they allow a user to train the agent through rewards and penalties, and they give a clear indication of the agent’s performance as defined by user enjoyment. If the agent is performing poorly, the users will click “dislike” fairly often, until a pleasing melody is generated. If the agent is doing a good job of predicting an aesthetically pleasing melody, the user will click “like” more often, resulting in a higher ratio of “likes” to “dislikes.”

As seen in the graphs below, that ratio improved over time. In the first 200 sessions with an untrained agent, the user clicked “like” for just over 45% of the notes offered. As the amateur agent continued to accrue data and modify its q-values, this percentage increased to nearly 55%.

In a “seeded” agent, one trained on a body of classical music before being subjected to user training, the ratio is higher still. By analyzing the transition patterns in these works, the seeding method supplied over 5000 pieces of training data which were considered positive feedback, allowing us to train an agent faster than was possible with limited and time-consuming human labor. When the seeded agent was activated, it received an initial like ratio around 62%. Even after the initial body automated training, the agent continued to improve with user feedback, topping a 65% success rate, an improvement of 41% over the untrained state.



It is important to note that the agent remains in "training mode" even as it sends improved melodies to the users. As it continues to train, the q-learning algorithm has an epsilon value of 0.1, meaning that it only follows its ideal policy 90% of the time. This causes the above statistics to be artificially decreased, as 10% of the time the action is completely random. Judging by the untrained state, random actions generate positive feedback ~40% of the time, or contribute approximately ~4%. Adding this to the 90% following the policy, we find that a perfect agent would only get positive feedback ~94% of the time given an optimal. In this light, an agent that gets 65.5% positive feedback is actually ~70% successful.



An unexpected trend discovered while analyzing the user feedback data is the change in negative feedback as related to melody length. As can be seen above, the percentage of "dislikes" increases as a melody grows longer. This can be explained by a user's natural creative response to a musical fragment. With a short melody, the user is more likely to give positive feedback and follow the agent in a random direction, whereas a longer melody establishes a certain style and phrasing, and a user is less tolerant of change as they begin to form their own ideas of how a melody should continue. As the melody length

increases, fewer and fewer notes will sound good with the entire melody, so the total percentage of dislikes will increase as the melody gets longer.

This trend shows that the agent's composition methods are more successful for short melodic phrases than longer melodies. In addition, it is clear that the "dislike" trend, while still increasing, is much lower in an agent with much experience. In a trained agent, the "like" percentage over the first five notes added to the start state averages higher than 75%. As our state model is based on phrases of 3 notes, it is logical that this model would break down as melody length increases.

Throughout the agent's training, the average length of the melodies generated before a user left the page or started over remained roughly constant, averaging at 10.17 notes. However, as seen in the graph above, the trained (seeded) agent was used to generate longer melodies. Although notes added to longer melodies eventually reach a nearly 100% "dislike" rate, it can be counted a success that as the agent improved, some users chose to remain with their melodies for longer periods of time, enjoying them enough to grow them to nearly ten times their initial length.

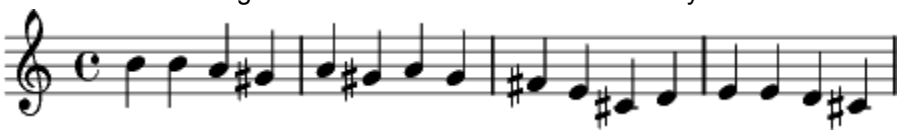
V. Discussion, Conclusions, and Future Directions

To model such a complex and creative problem as melody generation, we adopted a simplified approach. For the problem which we modeled, however – a q-learning algorithm of evenly spaced notes with limited intervals – our agent was highly successful. It excelled in particular at shorter melodic phrases (>75% positive feedback for two measures or less) and was able to generate four-measure melodies that were melodically cohesive and musically pleasing. By limiting our state model to three notes in length, we limited the length for which our melody generation would be viable, but by expanding the state length or adding a plurality of variable-length states, the model would hold for melodies of increased length.

The single most important open problem is how to add complexity to Melody Dragon to make it create more sophisticated melodies that satisfy more users. We came across many aspects of the project that can be improved to enhance the user experience, overall melody generation and the artificial intelligence.

The "Create a new Melody" button's largest handicap is that it restricts the number of notes in the melody to 16. This is problematic because we do not have any AI to determine what patterns fit with what number of notes. Thus, when a new melody is created with fixed length, even though it will have patterns that sound good, its ending is usually cut off by one or two notes.

For example, this melody sounds good overall but it would sound a lot better with two or three additional concluding notes added to the end of the melody:



One limitation of our user interface is that sometimes when a user wants a specific note, he/she will have to click the "dislike" button until that note appears. A good improvement wold be to allow users to add notes by adding a text field and a button that allows the user to enter a note in string format (C4, B#5 etc). Whenever a user uses this button, the new note is considered a transition that has been "liked".

Our results also showed that specifically because we used classical music, we obtained very specific patterns found in classical music, such as a gradual decrease in pitch. This could be avoided by having different pages that have been trained by different genre data sets. Since some users like specific genres and tonalities more, it would be useful to separate the webpage into genres with different data sets. By using patterns from different genres to train our agent, we could introduce more stylistic and tonal diversity to the melodies it generates.

Another improvement that would greatly increase the user feedback used to train our agent would be to add an undo button to the user interface. Whenever this button is used, the reward given to the undone transition is undone as well. Previous transitions may need to be cached for this. This will allow user feedback to be more accurate as it gives the users more flexibility in finding a melody that they really like,

or undoing a melody that they anticipated to be better. So, every user will be more certain of every “like” and “dislike”, which will increase the accuracy of our agent.

Introducing note duration would greatly increase the number of good melodies that can be produced by the agent, because a melody with correct note durations can sound a lot better than another melody with the same notes but different timings per note. Since note length can range from less than a 16th of a measure to multiple measures, we would need to restrict note lengths to a limited number of constants. We would also need to modify the user interface such that if someone likes the note but not the length of the note, they can specifically dislike the length but not the pitch of the note.

By expanding our musical state model to include variation of style, song length, note duration, and phrase elements (beginning, middle, conclusion) our learning agent would be able to remain conceptually simple while creating a wider range of more pleasing melodies.

Works Cited

[1] Conklin, D. (2003). Music generation from statistical models. In Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences (pp. 30–35). Brighton, UK: SSAISB.

[2] M. Allan. Harmonising Chorales in the Style of J.S. Bach. Master's thesis, School of Informatics, University of Edinburgh, 2002.

[3] V. Jiménez, A. Marzal, and J. Monné. A comparison of two exact algorithms for finding the N-best sentence hypotheses in continuous speech recognition. In Proc. 4th EUROSPEECH, pages 1071–1074, 1995.

[4] "Composing with genetic algorithms." Bruce Jacob. *Proc. International Computer Music Conference (ICMC '95)*, pp. 452-455. Banff Alberta, September 1995.