CSCE 790-002: Deep Reinforcement Learning and Search

Musical Deep Reinforcement Learning

Francisco Vilchez

December 11, 2020

Abstract

Computational musical composition tries to teach a computer what would be the best sequence of notes to play. This is an interesting task, since there is no specific process to measure whether one composition will sound better than another one; therefore, their quality will depend on each person musical preferences. In this project, we proposed using Deep Reinforcement Learning to capture the feedback from users during training so that way the computer can learn based on what the user consider is a good melody. Our project confirmed that the musical composition problem fits into a Reinforcement Learning problem. The computer was able to learn policies according to the feedback given by the user while it played its composition. The computer also stores the progress learned in order to continue learning later. Results and code of the project are available in Github[1].

Introduction

In this project we are trying to solve the automated musical composition problem using Deep Reinforcement Learning. Solving this problem with the usage of Reinforcement Learning becomes an interesting and innovative approach since it enables the interaction between the computer and humans for teaching which compositions sound good and that way improve the computer's compositional skills. This can become a difficult task to automate, since there is no specific way to measure if a composition is better or worse than another one. Music quality depends on each user's preferences, because of that, a composition that may sound nice for a group of people, may not sounds as nice as a different composition according to a different group of people's opinion. Based on this, we consider that it is important to create a bridge between computational musical composition and human feedback, reason why Reinforcement Learning can give promising results on this approach.

In this report, we will present the results that we have obtained by applying *Q-learning* and *Deep Q-learning*. We were able to teach the computer which notes are the best to be played after other notes according to the user's criteria by asking them for their feedback after a note or sequence of notes have been played. That way, we were able to generate a melody that match the user's preferences. We will give information about past works regarding musical encoding for computational composition and the usage of reinforcement learning in this area. After that, we will provide information about the approach used for solving this problem and explain the results obtained. Finally, we will mention future steps that we could take in this project.

Related Work

There have been different approaches for solving the musical composition problem including Evolutionary Algorithms, Markov Chains, Neural Networks and Reinforcement Learning, each of them have contributed in analyzing and trying to automate musical composition.

[1]Code available at: github.com/franciscovilchezv/eurydice.rl

The first approach analyzed during making this project was the one by Jaques et al. They proposed the usage of multiple neural networks to store the value for taking a possible action from a state and the usage of Deep Reinforcement Learning for keep improving those melodies. In their approach, they will start with an initial Neural Network which is trained from a sequence of melodies. Additionally, they will create an extra reward function based on music theory, that way, an score will be given to the action taken based on the criteria of music theory. The music theory reward works by making a list of different constraints and giving a score if that constraint was followed. The sum of all the constraints will be the final reward given by the music theory rules. On top of that, a Deep Reinforcement Algorithm will be used to improve the weights from the Neural Network created with historical data, by using the values from that Neural Network along with the reward given by the music theory rules. That way, their algorithm will have the freedom to create melodies which are based on its experience, but will also take into consideration the different musical rules that will "ensure" that their compositions have a pleasant sound. For all the neural networks involved in the training, *Recurrent Neural Networks (RNN)* were used [1].

The usage of *RNN*s for the musical composition problem has been corroborated by Eck and Schmidhuber, specifically with the usage of *Long Short Term Memory (LSTM)* RRNs. As they mentioned, *LSTM* is able to keep characteristics such as timing and musical structure while training, which was not possible with other type of models. They applied the solution to the *blues* musical genre and obtained pleasant results according to their work [2]. Both proposals based their work on optimizing the best note that should follow a previous sequence of notes, which is considered the main goal in the musical composition problem [3].

Our project differs from these two principal approaches in three aspects. Firstly, the usage of musical theory is completely ignored, since we consider that, even though musical theory can explain why a melody sounds nice, it should not constraint the set of notes that can be used [4] [5]. Secondly, we will try to introduce a novel data structure which is based on the note's grades instead of storing a note itself (more details in the next section). Finally, the principal goal of our project is to enable the human-computer interaction for the training process and that way we expect the computer to develop a more human-like musical skill. This is considered a vital point of the project, because the quality of the composition depends on each person's musical taste, because of that, by allowing the user to measure the quality of the generated compositions, the algorithm will learn how to play according the user's musical preferences.

Approach

As mentioned before, *Q-learning* and *Deep Q-learning* were applied in this project, each of them getting different results. In the next sections we will describe each of the components in our Reinforcement Learning algorithm applied to the Musical composition environment. Additionally, we will detail the constraints that were applied in the project in order to reduce its scope.

State

The state in out Reinforcement Learning model is defined by the sequence of notes that have been player so far. For example, an starting state can be seen in Figure 1.

If we decide to play another note after it, then the new state will be represented by both notes (all the notes played so far). We can see an example in Figure 2.

However, a *note* does not only refer to the pitch that is played. It involves another criteria. For example:

- Note frequency or name
- Duration

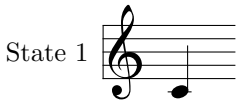

Figure 1: Initial state represented by playing the note C

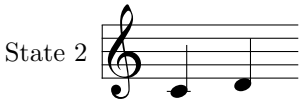

Figure 2: State represented by playing the note D in the State 1

- Chord in which the note was played
- Tonality of the composition in which the melody was played

Because of that, we will provide more information about what information about the note we are using in its representation and the way it is being encoded in our algorithm.

Note representation

Numerical notation is a representation for notes that has been widely used in previous works [5]. The idea is to use a specific number to represent each of the notes. However, using a number for each note will constraint the learning process to the specific characteristics of the current environment, like the tonality in which the model is training, or the harmonic sequence; because of that, we are suggesting the usage of a relative value of the note according to the harmony and tonality of the composition. For example, we can see the encoding for one note in Figure 3.

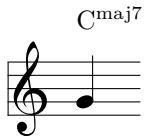

Figure 3: A G note which is represented by 5 in a C major tonality

However, the same note will have a different encoding if we are in a different tonality, as we can see in Figure 4.

At the time of writing this report, only pitch is being taking into consideration in the note encoding. Updates regarding the encoding of notes and other possible changes are visible in the project's release page[2].

Goal State

Compositions are usually defined by a specific number of *bar*s that we need to compose in. For example, a jazz soloist will may have to immediately compose a melody that fits into 16 musical *bar*s during a jam session. If we know the *time signature* of the composition we are composing on, and the number of musical bars, then we can know how many musical times we need to complete

[2]Releases available at: github.com/franciscovilchezv/eurydice.rl/releases

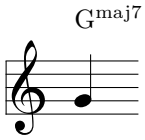

Figure 4: Same *G* note is now represented by *1* in a G major tonality

and based on that determine if we reached our goal state. At the time of writing this report, the goal state was defined by a fixed number of notes in an hyperparameter variable.

Q-learning

In this project *Q-learning* was used as our reinforcement learning algorithm. Since the number of states is determined by the possible amount of note combinations that we can make, our total amount of states can be too large for storing it in memory. Because of that, Q-learning allows us to learn base on experience and thus, only store values experienced. We will give more detail about the components in our Q-learning algorithm in this section.

Actions

As we mentioned, the action is the possible note that can be played based on our current state. Without getting into any musical theory detail, we could say that any note could follow any other note. Because of that, the range of possible actions is also very broad. In order to simplify the scope of our project, for now we are only dealing with notes from the C Major tonality ranged in one musical scale. In other words, the notes C4 D4 E4 F4 G4 A4 B4 C5 as shown in Figure 5.

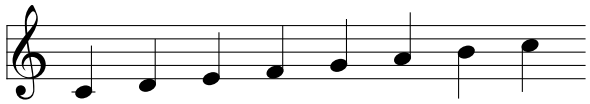

Figure 5: Notes available as a possible action.

However, this restriction will be ignored in the future since we should not limit the possible notes that we can use. ϵ-greedy was used as our exploration method in our Q-learning algorithm.

Transition Reward

The given reward by choosing an action over another is determined by the feedback that the user gives for the melody created so far. Each time an action is selected, the program asks the user to qualify it as:

Good If the composition is pleasant so far and the user would like to hear it in future occasions. Positive reward given.

Neutral If the melody does not have a pleasant or unpleasant sound so far, so it is not possible to decide yet if you like it or not. No reward given.

Bad If the melody sounds terrible and the algorithm should avoid to play that action in the previous state. Negative reward given.

During the development of this project we noticed that giving manual rewards would delay the initial steps in the creation of the algorithm since it takes a lot of time because the user needs to listen to every melody generated and give a reward based on that. Due to this reason, an automated reward was included for testing purposes, in which a basic musical criteria is followed for giving the rewards. For example, rewarding positively only if the notes are in a descending order.

Q-table and Neural Networks

Rewards for each action taken from a specific state were initially stored in a Q-table. After confirming it was working as expected, we replaced it with Neural Networks in order to try to improve the results. According previous works, it is considered that Recurrent Neural Networks provide the best results in the area of musical composition, however, we started with Neural Networks and we are considering changing that into Recurrent Neural Networks in a future. The Neural Network has X units as input, where X is equal of the number of notes that will determine a Goal State in our algorithm. During testing, we limited the number of notes that we can use to a total of 8 notes. The numerical note values are used as input of the neural network.

Experimental Results

In this project we were able to teach the computer what sequence of notes would have a pleasant sound according to our criteria, and by applying a greedy search the computer was able to generate those results successfully. As mentioned in the previous section, in this project we used Q-learning and Deep Q-learning. Each of them gave us different results. We will show the results and discuss them in this section. The information regarding how to run each of the functionalities is available in the Github README file[3]

Q-learning

Q-learning (storing results in a Q-table) provided the most accurate results for our project. The environment was adapted for testing purposes to only have one optimal policy and the Q-learning algorithm was able to find it. In order to speed up the testing process, an automated reward was generated. In this section we will discuss the results for the automated reward and the human-computer policy.

Automated reward

A function that rewarded playing a note lower than the previous one was created in order to speed up testing during development.

In musical terms, what this is trying to teach is a descending scale. We gave a -100 reward in case the algorithm played a note higher than the previous one, and a $+10$ if it played a note lower than the previous one. We could see that our algorithm was able to learn the optimal policy after 1500 iterations in average.

As we can see in Figure 1, the optimal policy was not found yet, since the algorithm is still deciding to use a higher note in the position 6 than the previous note (position 5).

```
Q-learning, episode 499
[<Note.C5: 7>, <Note.B4: 6>, <Note.G4: 4>, <Note.E4: 2>, <Note.C4: 0>, <Note.D4: 1>,
<Note.D4: 1>, <Note.E4: 2>]
Playing C5 (523.25 Hz) for 0.4s
```

[3]README file with code details: github.com/franciscovilchezv/eurydice.rl/blob/main/README.md

```
Playing B4 (493.88 Hz) for 0.4s
Playing G4 (392.00 Hz) for 0.4s
Playing E4 (329.63 Hz) for 0.4s
Playing C4 (261.63 Hz) for 0.4s
Playing D4 (293.66 Hz) for 0.4s
Playing D4 (293.66 Hz) for 0.4s
Playing E4 (329.63 Hz) for 0.4s
```

Listing 1: Iteration 500: Not efficient policy due to note transitions like C4 to D4 in position 5.

In Listing 2 we notice that the algorithm finally found the optimal policy for the scenario that we proposed (each note should be lower than the previous one). Of course, this condition does not have any musical validity, it was only used for testing the learning capability of our algorithm. After this goal was accomplished, we felt confidence enough to start trying the algorithm with human interaction.

```
Q-learning, episode 999
[<Note.C5: 7>, <Note.B4: 6>, <Note.A4: 5>, <Note.G4: 4>, <Note.F4: 3>, <Note.E4: 2>,
<Note.D4: 1>, <Note.C4: 0>]
Playing C5 (523.25 Hz) for 0.4s
Playing B4 (493.88 Hz) for 0.4s
Playing A4 (440.00 Hz) for 0.4s
Playing G4 (392.00 Hz) for 0.4s
Playing F4 (349.23 Hz) for 0.4s
Playing E4 (329.63 Hz) for 0.4s
Playing D4 (293.66 Hz) for 0.4s
Playing C4 (261.63 Hz) for 0.4s
```

Listing 2: Iteration 1000: Optimal policy. Each note is lower than the previous one.

Human interaction training

For allowing an easy training after an action was taken, our project displays the notes generated so far and reproduces its sound. After that, the user is able to qualify the composition as *Good*, *Bad* or *Neutral*. The process explained looks as shown in Listing 3.

```
> python run_music_generation.py --step 100 --episodes 200 --interactive_mode
  No model specified in --model argument. Training wont be saved.
  q-learning
  Q-learning, episode 0
  Playing C5 (523.25 Hz) for 0.4s
  Type your feedback good(g), bad(b), neutral(n), stop(s): g
```

Listing 3: Human training for the first transition in an episode

We were also able to teach a more interesting musical phrase which is shown in Figure 6 using the human interaction training.

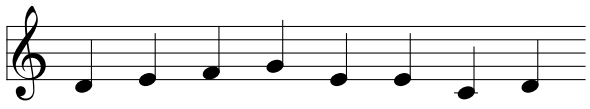

Figure 6: A variation of "The Lick". A famous Jazz cliche

We can see the result by using the `--results` option, which will run a greedy search and return the best policy that has been found so far as shown in Listing 4.

```
> python run_music_generation.py --model thelick --results
  Model 'trainings/thelick.pkl' loaded
  [<Note.D4: 1>, <Note.E4: 2>, <Note.F4: 3>, <Note.G4: 4>, <Note.E4: 2>, <Note.E4: 2>,
```

```
<Note.C4: 0>, <Note.D4: 1>]
Playing D4 (293.66 Hz) for 0.4s
Playing E4 (329.63 Hz) for 0.4s
Playing F4 (349.23 Hz) for 0.4s
Playing G4 (392.00 Hz) for 0.4s
Playing E4 (329.63 Hz) for 0.4s
Playing E4 (329.63 Hz) for 0.4s
Playing C4 (261.63 Hz) for 0.4s
Playing D4 (293.66 Hz) for 0.4s
```

Listing 4: Playing the optimal policy in the model *thelick*

Even though the algorithm is correctly learning values for notes, it is difficult to try to teach a second melody and store it in the q-table since the algorithm will act ϵ-greedily. In order to try to store multiple melodies, a different exploration method must be used, otherwise we will always play either the optimal melody during training, or a completely random melody. Additionally, the results should be printed by randomly selecting from the best policies and not the best policy most of the times, in order to get some variation in the melodies generated.

The process followed for training "the lick" can be found in *thelick.training* log file in Github[4].

Deep Q-learning

After getting good results with Q-learning, we decided to expand it and test its behavior using a Neural Network for storing the values instead of the Q-table. It was executed with the parameters shown in Listing 5.

```
> python run_music_generation_py --aprox_q_learning --step 500 --episodes 5000 --epsilon 0.3
  Q-learning, episode 0
  Q-learning, episode 1
```

Listing 5: Running Deep Q-learning.

However, the results were not as good as expected. After running the algorithm with the automated reward in the testing environment with one optimal policy possible, the Deep Q-learning algorithm was not able to find the optimal policy, as we can see in Listing 6.

```
Q-learning, episode 4999
[<Note.C5: 7>, <Note.B4: 6>, <Note.F4: 3>, <Note.E4: 2>, <Note.C4: 0>, <Note.C5: 7>,
<Note.A4: 5>, <Note.C4: 0>]
Playing C5 (523.25 Hz) for 0.4s
Playing B4 (493.88 Hz) for 0.4s
Playing F4 (349.23 Hz) for 0.4s
Playing E4 (329.63 Hz) for 0.4s
Playing C4 (261.63 Hz) for 0.4s
Playing C5 (523.25 Hz) for 0.4s
Playing A4 (440.00 Hz) for 0.4s
Playing C4 (261.63 Hz) for 0.4s
```

Listing 6: Iteration 4999 with Deep Q-learning: Not the optimal policy.

We consider that the conversion of our state to the input of the neural network may not be the best, because of that, we are not getting the results that we were expecting. Additionally, the usage of RNN as mentioned in the state of art, could possibly lead to better results. Improvements and trying different methods for Deep Q-learning are still needed in order to continue expanding its capabilities.

Previous works in the area got different results since they are relying on the usage of music theory, which is what we are trying to avoid in the training process since we consider it will provide more *natural* results. Improvements in this project need to be done in order to be competitive with results from the current state of art.

[4]The Lick log: github.com/franciscovilchezv/eurydice.rl/blob/main/source/trainings/thelick.training

Conclusion

In this project we were able to apply Q-learning and Deep Q-learning in order to teach the computer an optimal policy for a sequence of musical notes. We created a note encoding method that allowed our algorithm to correctly manipulate musical notes, and proposed ideas for better encoding methods using relative values for improving this results. We defined and developed the components of a Reinforcement Learning algorithm in the musical composition environment. We create an automated reward process for testing and development purposes, and after that, implemented the human-computer reward interaction process. A process for reproducing the results and getting feedback from the user was created in order to allow the fluid training process. We successfully applied Q-learning for learning an optimal policy in our musical environment, and later applied Deep Q-learning not getting as good results as with the plain Q-learning.

We consider that improvements are still needed in the notes encoding in order to start storing for information about the notes, e.g. duration, tonality, harmonic progression, dynamics, instrument, etc. Additionally, more investigation is needed in the usage of Neural Networks for storing the transition values, since other techniques such as Recurrent Neural Networks could provide better results that the ones obtained. Finally, we consider that we could try to usage of previously validated melodies in order to speed up the training process and create some diversification of musical compositions on which we can base our training.

References

[1] Natasha Jaques, Shixiang Gu, Richard E. Turner, and Douglas Eck. Generating music by fine-tuning recurrent neural networks with reinforcement learning. In *Deep Reinforcement Learning Workshop, NIPS*, 2016.

[2] Douglas Eck and Juergen Schmidhuber. Finding temporal structure in music: Blues improvisation with lstm recurrent networks. In *Proceedings of the 12th IEEE workshop on neural networks for signal processing*, pages 747–756. IEEE, 2002.

[3] P. Todd and G. Loy. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13:173–194, 1989.

[4] Francisco Vílchez, Jose Astuvilca Fuster, and César Beltrán Castanón. Artificial musical pattern generation with genetic algorithms. In *2015 Latin America Congress on Computational Intelligence (LA-CCI)*, pages 1–5. IEEE, 2015.

[5] John A Biles. Performing with technology: Lessons learned from the genjam project. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*. Citeseer, 2013.