A Generative Model for Creating Musical Rhythms with Deep Reinforcement Learning

Seyed Mojtaba Karbasi[1,2], Halvor Sogn Haug[2], Mia-Katrin Kvalsund[2], Michael Joseph Krzyzaniak[1,2], and Jim Torresen[1,2] *

[1] RITMO Centre for Interdisciplinary Studies in Rhythm, Time, and Motion,
University of Oslo, Norway
[2] Department of Informatics, University of Oslo, Norway
{mojtabak, halvorsh, mkkvalsu, michakrz, jimtoer}@uio.no

Abstract. Musical Rhythms can be modeled in different ways. Usually the models rely on certain temporal divisions and time discretization. We have proposed a generative model based on Deep Reinforcement Learning (Deep RL) that can learn musical rhythmic patterns without defining temporal structures in advance. In this work we have used the Dr. Squiggles platform, which is an interactive robotic system that generates musical rhythms via interaction, to train a Deep RL agent. The goal of the agent is to learn the rhythmic behavior from an environment with high temporal resolution, and without defining any basic rhythmic pattern for the agent. This means that the agent is supposed to learn rhythmic behavior in an approximated continuous space just via interaction with other rhythmic agents. The results show significant adaptability from the agent and great potential for RL-based models to be used as creative algorithms in musical and creativity applications.[3]

Keywords: Deep Reinforcement Learning, Rhythm Generation, Generative Model, Interactive Robots

1 Introduction

Creating music by AI is a growing topic in music technology and computer science in recent years. The significant capabilities of the Deep Neural Networks in pattern recognition, combined with Reinforcement Learning (RL) algorithms, attracted many researchers to utilize them for developing generative models. Are such generative models able to learn to create musical rhythms by interacting with other musical agents?

RL algorithms can learn to obtain a goal by interacting with an environment. Deep reinforcement learning (Deep RL) is a method for training an agent to perform certain actions given certain states. The agent consists of a neural net

* This work is partly supported by the Research Council of Norway through its Centres of Excellence scheme, project number: 262762.

[3] The source codes of this work can be accessed through https://github.com/mojtabak-rob/RLSquiggles

that learns a policy to take desired actions (Sutton & Barto, 2018). After it was first introduced, Deep RL has been used for various tasks and achieved human level results (Mnih et al., 2015; Lillicrap et al., 2015; Arulkumaran, Deisenroth, Brundage, & Bharath, 2017; Mnih et al., 2013). To answer if rhythmic patterns can emerge in robot systems through interaction, the system is required to take actions that makes us perceive it as rhythmic. Deep RL is a good tool to make an agent learn to take actions given a state, especially when the states may be from an infinite set. The set of possible states will change according to the agent's actions, and the other robot agents within the environment.

There have been significant works into music technology, and into the field of creating music, trying to develop algorithms for rhythm generation. Within the area of creating rhythm, Kaliakatsos-Papakostas, et. al have used Evolutionary algorithms to create rhythmic patterns by deriving them from a base beat (Kaliakatsos-Papakostas, Floros, & Vrahatis, 2013). Markis et. al have taken advantage of combining a Recurrent Neural Network with a Feed-Forwards Neural Network, to generate a drum beat based on a bass line (Makris, Kaliakatsos-Papakostas, Karydis, & Kermanidis, 2017). There have also been previous works using Reinforcement Learning-based algorithms for Music Generation (Lan, Tørresen, & Jensenius, 2019; Kotecha, 2018; Fryen, Eppe, Nguyen, Gerkmann, & Wermter, 2020; Collins, 2008; Le Groux & Verschure, 2010; Derbinsky & Essl, 2012; Assayag, Bloch, Chemillier, Cont, & Dubnov, 2006). Common for these approaches is that they give their architecture a pre-existing idea of what the rhythm should consist of, in the form of a discrete output that is made to fit 4/4 bars. A goal would be to create an architecture that doesn't require this knowledge, and that could generate a beat in a practically continuous environment.

The main contribution of this work is to introduce a generative model based on Deep Reinforcement Learning which can learn to create musical rhythms by interacting with other musical agents. The model learns rhythmic patterns from scratch via interaction with other musical agents. The key question is if any rhythmic pattern can emerge in a robotic system just through interaction with other musical agents, without having any predefined rhythmic patterns.

2 Method

2.1 Dr. Squiggles Platform

Dr. Squiggles is an interactive robot that generates rhythmic patterns by tapping (Krzyżaniak, 2021). It listens to rhythms played by humans or other robots, and attempts to play along. The robots can also listen to each other and play together. The robots use a histogram-based algorithm for generating rhythm by dividing time intervals into 16 parts and calculating the probability of playing notes in each part. In this work, we use a simulated Dr. Squiggles platform as the environment for our agent to learn rhythmic patterns. In this setting, we use two Dr. Squiggles robots with a histogram-based algorithm listening to each other and the agent. The agent will listen to one of the robots and try to follow its rhythm using the Deep Reinforcement Learning model (Figure 1).

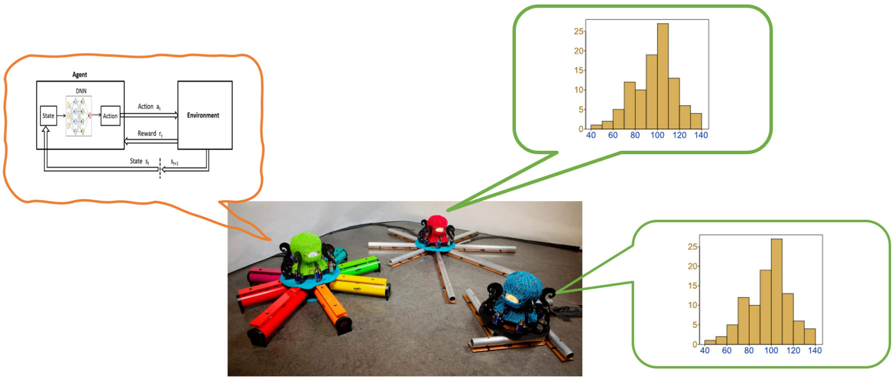

Fig. 1. Dr. Squiggles robots setting. We have used two Dr. Squiggles robots with a histogram-based rhythm generating algorithm in the environment. The agent is the third robot and uses a DRL algorithm to learn the rhythmic patterns from the environment.

2.2 Deep Reinforcement Learning Model

A Deep RL algorithm is based on ordinary RL methods which receive rewards from the environment after each action and learns to maximize the expected cumulative rewards in the long term. In Deep RL, a Deep Neural Network is used to estimate the expected reward (or value) and is updated in each trial (Sutton & Barto, 2018; Arulkumaran et al., 2017).

The algorithm consists of four parts: The agent, the environment, the observation space, and the rewards, which will be described in the sections below:

The Agent For the agent, a Deep Q-Learning Network (DQN) is used. This consists of an epsilon-greedy policy and a rather small network, used to determine the action: To tap or not to tap. That is, the output of the agent is either 0 for not tapping, or 1 for tapping. The agent plays the role of the third Dr. Squiggles robot shown in Figure 1, by using a DRL algorithm for playing.

The choice of using a DQN was motivated by having the agent able to learn the policy having a continuous observation space. We used a feed-forward network in this work consisting of five hidden layers. A small network is essential for avoiding overfitting.

The Environment The environment can be any kind of musical performer that generates rhythm. We used two Dr. Squiggles as the environment for the agent. For modeling the environment, we should define the state and reward signals. These are the feedback signals that the agent will use to be trained and also make actions. They are described in their own sections below.

The DRL concepts related to the environment are step, episode, reward, observation space and action space. They are defined as follows:

1. Step: Every 10 ms, the agent chooses an action that is executed within the environment. A reward is calculated and sent back, completing one step. 10 ms was chosen as it is beyond the scope of human reaction, making the system effectively continuous.
2. Episode: An episode is a set number of steps before the environment terminates, and resets itself. In this project's environment, an episode is 1000 steps, or 10000 ms / 10 seconds, but the agent will be able to play for any given length when fully trained. The reason for choosing 10 seconds is that at the start of each episode the tempo is randomized, so having the episode be 10 seconds allows the agent to be trained on many different tempos.
3. Reward: In each step, the agent is given a reward based on its action and previous environment state, that is used to update the agent's policy. This is described in detail in Section 2.2 below.
4. Observation space: The current state representation given to the agent for it to choose an action to perform. This is described in detail in Section 2.2 below.
5. Action space: The set of actions the agent can perform. Here, the set contains the actions 1 and 0, 1: Tapping, 0: Not tapping.

For each episode, the environment starts off with a randomized value for the time between each 16th note, from 100ms to 200ms. This is done to give it a random tempo. The environment is started with random histograms for each Dr. Squiggles.

The reason for having a random tempo is to make the agent more dynamic, so it's not limited to having to follow along with one specific tempo. More importantly, it is crucial that the agent does not train to become a neural network that detects when one input is e.g. 20, but instead actually observes the time between each onset by itself and adjusts accordingly. If it can do this, it would help prove that it was synchronizing to the Dr. Squiggles in the environment, and playing rhythmically because of their behavior.

Observation space For generating rhythms, time-based and frequency-based features can be considered (Sethares, 2007). In our setting we want the agent to learn rhythmic patterns in real time. Time-based features can be used more effectively in real time applications since the agent can not access the whole time history of the rhythms in advance to extract the frequency-based features. Although it is possible to use frequency based features in real time, using time-based features is faster and demands less computational complexity. A simple time-based feature that can be used in each step can be the time that has passed since each of the last n times the robot in the environment has played. Here, n can be any arbitrary number, but with a larger n we will need a larger network and it makes the model more complex. By using these states, the agent is looking to the previous events and uses some kind of memory. Using memory is essential for rhythm generation because at each moment playing a note or not depends on previously played notes and their timing.

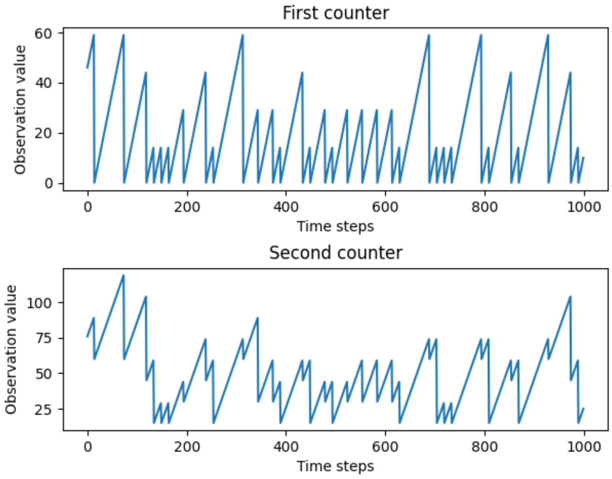

Fig. 2. A visualization of the observed states for one episode.

Figure 2 shows an example of the observed states in an episode by considering the 2 last notes. In addition to the basic counters from when the environment last played, there have also been attempts with having *n-1* states that represent the time difference between each of the states. This would allow the agent to get knowledge of the temporal structure of the environment, that could be used to determine when to play.

The Rewards Defining the reward function is the key for the algorithm to learn the behavior that is desired. The goal for the agent is to learn rhythmic behavior from the environment. One way is to give rewards when the agent is playing at the same time as the environment, however in this case the agent only learns to repeat the notes. Here, we introduce a different strategy to define the reward function. The agent receives positive reward when it mirrors the last two notes played by the environment. In this case the rhythmic patterns played by the agent and the environment can create new rhythmic behavior when playing together. The reward function is explained in more detail in Appendix A.

3 Result

The model and the environment are simulated using the *TensorFlow* library. The network used for training in each case consists of five linear layers with twenty nodes each. The agent is trained for 20000 time steps in each case, randomly selected from a replay buffer.

The agent shows a rhythmic pattern that follows the rhythm of the environment, and does not just play when the environment is playing. However, it also

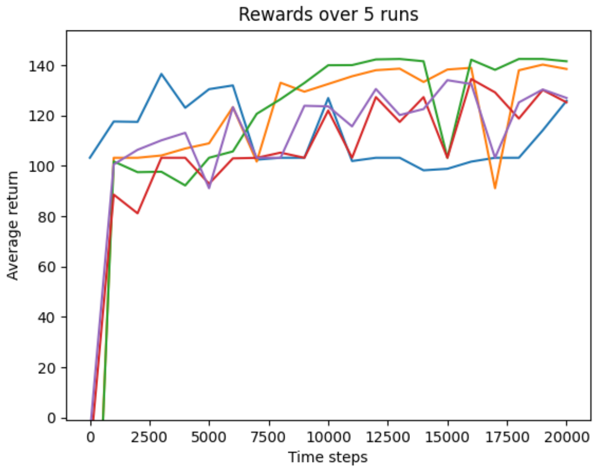

Fig. 3. The average rewards during training for 5 different runs.

only plays one note after the environment stops playing, and then gets lost. This result was achieved by giving a reward for when it mirrored the environment and when it otherwise stayed quiet, as well as punishments for when it missed a mirrored onset and tapped when it should be quiet.

The training is run for 20000 time steps and the agent can converge with an appropriate definition for the reward function. Figure 3 illustrates the average cumulative reward per episode collected during training, and it eventually converging to a stable pattern that gets similar amounts of reward.

Several other reward functions were tried that did not work. Specifically, any reward based on actions taken in the past were not effective, e.g. giving negative rewards based on playing the same note continuously/not playing for a long time. It seems the agent is not able to connect the reason for this punishment, even when given how long ago it played itself. Additionally, these punishments can be inevitable to encounter/avoid when the random chance of an epsilon greedy exploration is involved, as the agent cannot 'restrain' itself to the action it chooses. Therefore it might work better if on-policy learning is used, but that might risk the agent converging early, as it can't explore.

As for the states, having the steps since the environment has played and the difference between when the environment played seems to have been helpful, and let the agent get some idea of the rhythm. n has been kept small, and there does not appear to be a huge benefit to increasing the number of n. As for feeding the agent's output into itself the next step, even though it was included, does not seem to have had a huge effect compared to previous experiments. Lastly, giving the agent the number of steps since it last played does not appear to have been very effective, for the same reason giving punishments due to playing

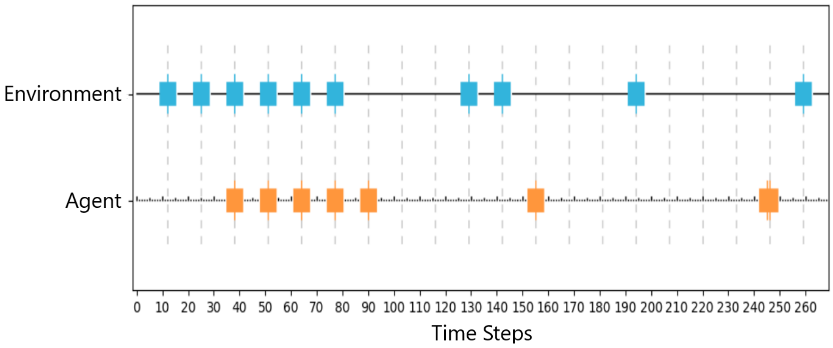

Fig. 4. The output of the agent and the environment after training. The blue and orange squares indicate tapping by the environment and the agent respectively. The dashed lines indicate the time divisions according to the histogram-based algorithm (a video sample is available here: https://youtu.be/GSArvObZxeU)

too much/too little, in that it's often dodged in training, so it is not properly considered before validation.

Additionally, we trained standard classifiers on a balanced data set of labeled observations and saw what we could achieve with that approach. The classifiers were tuned through a grid-search where they were evaluated on a score equal to the lowest class precision. The score was chosen to help the classifiers get the right balance between the labels.

Using classifiers to learn the observation space did not help to converge to any rhythmic behavior. In table 1, the performance on an episode of the agent and the four classifiers can be seen, where they were both evaluated on lowest class precision. The agent frequently gets full score, while the classifiers deliver unreliable and low-scoring performances. In terms of behavior, the classifiers were too noisy, but showed promise to learn the mirroring pattern.

Apart from the complexity of the data, the dynamic behavior of the environment which responds to the action of the agent and changes the rhythmic pattern according to what it hears from the agent makes it almost impossible for the classifiers to learn the rhythmic behavior. This indicates why an interactive algorithm, like the Deep RL model we used, is needed for this problem. A Deep RL model also lets the model avoid the pitfalls of an unbalanced data set, as the rewards and low learning rate allows the agent to escape from the local maxima of classifying everything as a 0 or 1.

4 Discussion

The results show that the Deep Reinforcement Learning model can be effectively used for training an agent to generate musical rhythms through interaction. The

Table 1. Table of lowest class precision for sci-kit Random Forest, Multi-Layer Perceptron, K-Nearest Neighbor and a Support Vector Classifier. All classifiers fit to a balanced data set of 40000 labeled observations.

Trial	Agent	RFC	MLP	KNN	SVC
1	1.0	0.4473	0.3245	0.0	0.5404
2	0.6	0.3333	0.5421	0.536	0.3333
3	1.0	0.1053	0.369	0.1053	0.5219
4	0.9888	0.0	0.5333	0.5939	0.5333
5	1.0	0.2	0.4154	0.2	0.4338
6	1.0	0.0	0.3241	0.4796	0.5245
7	1.0	0.1594	0.1222	0.411	0.2246
8	1.0	0.0	0.2775	0.0	0.3422
9	0.999	0.0	0.4618	0.5056	0.5861
10	1.0	0.3722	0.1526	0.0	0.3804

flexibility of the Deep RL algorithm enables the agent to learn musical rhythms without pre-programming the rhythmic patterns or discretizing the temporal features or action space. This is the main advantage of this approach which is not limited to certain defined rhythms.

Additionally, the algorithm is based on interaction with the environment and is helpful when we have an interaction between humans and robots. This approach makes the robots adaptive to the environment and more creative and maybe more entertaining for humans.

This algorithm can be improved and expanded in different ways. One important thing to be improved more precisely is the reward function.

An interesting expansion of the agent would be to add in an LSTM-layer, as this would allow the agent to retain an idea of the overall structure of what it is playing, which could be beneficial when it comes to producing an overarching rhythmic structure. The RNN component might be helpful to model the temporal dynamics of the environment with more complexity. It could also allow the agent to keep playing without depending on the environment for an input. However, the training of the RNN networks takes more time and using it in real-time interactive applications is a big challenge.

5 Conclusion

A musical agent trained with Reinforcement Learning was able to adapt to rhythmic behavior with no primary knowledge about the characteristics of the musical rhythm. The results show that the model is able to find the basic rhythmic patterns from the environment. However, the rhythm played was still quite simple, and the agent seems to be blind to the dynamical behavior of the environment which is a complex task. There is further work to be done, both in expanding the complexity of the agent, and in creating better reward functions that can lead to more creative rhythms.

References

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, *34*(6), 26–38.

Assayag, G., Bloch, G., Chemillier, M., Cont, A., & Dubnov, S. (2006). Omax brothers: a dynamic yopology of agents for improvization learning. In *Proceedings of the 1st acm workshop on audio and music computing multimedia* (pp. 125–132).

Collins, N. (2008). Reinforcement learning for live musical agents. In *Icmc.*

Derbinsky, N., & Essl, G. (2012). Exploring reinforcement learning for mobile percussive collaboration. In *Nime.*

Fryen, T., Eppe, M., Nguyen, P. D., Gerkmann, T., & Wermter, S. (2020). Reinforcement learning with time-dependent goals for robotic musicians. *arXiv preprint arXiv:2011.05715*.

Kaliakatsos-Papakostas, M. A., Floros, A., & Vrahatis, M. N. (2013). Evodrummer: Deriving rhythmic patterns through interactive genetic algorithms. In *International conference on evolutionary and biologically inspired music and art* (pp. 25–36).

Kotecha, N. (2018). Bach2bach: Generating music using a deep reinforcement learning approach. *arXiv preprint arXiv:1812.01060*.

Krzyżaniak, M. (2021). Musical robot swarms, timing, and equilibria. *Journal of New Music Research*, 1–19.

Lan, Q., Tørresen, J., & Jensenius, A. R. (2019). Raveforce: A deep reinforcement learning environment for music. In *Proceedings of the smc conferences* (pp. 217–222).

Le Groux, S., & Verschure, P. (2010). Towards adaptive music generation by reinforcement learning of musical tension. In *proceedings of the 6th sound and music conference, barcelona, spain* (Vol. 134).

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Makris, D., Kaliakatsos-Papakostas, M., Karydis, I., & Kermanidis, K. L. (2017). Combining lstm and feed forward neural networks for conditional rhythm composition. In *International conference on engineering applications of neural networks* (pp. 570–582).

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Sethares, W. A. (2007). *Rhythm and transforms.* Springer Science & Business Media.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

A Reward Function

Here is the detailed reward function we used:

$$r(s,a) = \begin{cases} 2/t & \text{if } a{=}0 \text{ and } \textit{s(3)} \neq \textit{s(1)+1}\text{, good silence} \\ 1/t & \text{if } a{=}1 \text{ and } \textit{s(3)} = \textit{s(1)+1}\text{, mirroring last two notes} \\ -1.57(t-1)/t & \text{if } a{=}0 \text{ and } \textit{s(3)} = \textit{s(1)+1}\text{, should be mirroring} \\ -1.85/t & \text{if } a{=}1 \text{ and } \textit{s(3)} \neq \textit{s(1)+1}\text{, plays off beat} \end{cases}$$

In this function a is the agent's action and t is the episode's allowed minimum time between the environment making sound. s is the observed state, *s(3)* is the time difference between the last two notes played by the environment, and *s(1)* is the time since the environment last produced an onset. Note that the agent must predict a step ahead.

It is especially important to get the ratio of the rewards and punishments right, so that the agent could not simply stay silent and receive more reward than if it played, or vice versa. Additionally, the tempo would change between each episode, making it crucial that the balance was scaled in such a way that each episode would have the same need to make noise on the beat and otherwise stay quiet to get the best cumulative reward. This is why the rewards are scaled with t.

Another issue is that the agent's action space is not constrained in any way, making it prefer to beep continuously instead of tapping once. The desired behavior is rhythmic, requiring it to tap sparingly, and so the balance of rewards between silent and not silent is used to achieve single taps. It might be the case that an agent that has constraints on its actions would produce more rhythmic behavior.