JUST BEAT IT: CREATING PERCUSSION COMPOSITIONS USING DEEP Q-LEARNING TO TUNE NEURAL NETWORKS

A PREPRINT

Sriram Ravula [*]
sr39533
sriram.ravula@utexas.edu

Richard Cao
ryc247
richardcao@utexas.edu

December 10, 2019

ABSTRACT

Supervised learning using recurrent neural networks (RNNs) or convolutional neural networks (CNNs) are common ways to predict time-series. However, it's known that they are not ideal for determining long-distance or global temporal structure. Google's work with Magenta uses Reinforcement learning to apply a set of rules on generated music to learn an underlying structure and produce piano compositions that sound pleasing. In this paper, we extend this approach to train a supervised model on drum compositions, then tune the model using deep Q-learning to improve the musicality of the drum compositions produced by the model. We find that, possibly due to some suboptimal decisions regarding state and action representation and model training, using a moderately exploratory policy during action selection produces the best RL drum beat generation model, but that a supervised model still outperforms RL models for producing pleasing-sounding compositions. Our project code is found at: `https://github.com/Sriram-Ravula/RL-Music-Tuner`, and our video can be found at: `https://youtu.be/I777JsSk9Pw`.

1 Introduction

Music generation problems are typically solved using Long Short-Term Memory (LSTM) networks, which are able to capture long-distance temporal relationships in time-series. However, a generative model trained with supervised learning is usually not sufficient for making music that is both novel and pleasing to the ear. Google's work [1] used reinforcement learning to supplement the music generation model to make better music compositions. Their music generation model was trained on an expert-made data set of music tracks, using supervised learning. Our project is based on Google's work, with one key difference - we are generating drum beats instead of traditional melodies on a piano. We took this direction because of our backgrounds in playing rhythm-heavy instruments like tabla and drums and interest in hip-hop production. The challenge with our drum beat generation is that percussive sounds are more or less independent of each other, whereas musical notes have a relationship in terms of pitch distance. However, each percussive instrument can be treated as a separate channel, with two possible options - on or off. We will design our system based on these changes.

2 Related Works

The initial inspiration for our work comes from [1], in which the authors train an LSTM to generate piano notes sequentially, then use the weights of the network to initialize a deep Q-network which is finally optimized using custom music theory rewards in its loss formulation. Our work is similar in that we plan to use a trained neural network to get an action probability given a state, then use that same network as the initialization for our deep Q-network. However, due to the difficulty of custom reward engineering for music theory, we plan to simply use supervised inverse reinforcement learning to learn a suitable reward function. In addition, our states and actions consist of different percussion instruments instead of notes on a piano, so we adopt a new state and action space for our problem.

[*] `https://github.com/Sriram-Ravula/RL-Music-Tuner`, `https://youtu.be/I777JsSk9Pw`

Another use of LSTMs is in the work of Eck et. al. They apply LSTMs to the problem of music generation to demonstrate that they are successful in learning temporal structure of music, even for temporally distant events [2]. In the case of drum beat generation, the "chords" will be various combinations of percussive sounds to achieve a certain effect, and the melody will be a progression of generated sounds in relation to previous sounds. Since long-distance relationships between notes won't be as present in the percussive aspect of drum beats, especially since each individual drum instrument can be treated as a different channel, this work will primarily be used for generation of the single melodic line. Although our project differs in terms of melody vs. percussion, the advantages of LSTMs are still applicable.

As our problem requires a learned reward function based on labeled data, we will have to use an inverse RL approach, similar to the apprenticeship learning algorithms put forth by Abbeel and Ng [3]. The authors seek to find a policy that optimizes an unknown reward function by imitating the actions taken by an expert from given trajectories. They give an algorithm that provably finds a policy that is ϵ-close to the expert policy, based on the discounted trajectory of state features. While our problem requires inverse learning from labeled data, we seek the reward function itself, not a policy that mimics an expert. Nonetheless, the approach that the authors take indeed generates a candidate reward function, based on maximizing the margin between the expert policy and a candidate policy, which we could potentially use in our deep Q-network's loss formulation.

Our problem requires us to use a learned expert reward function which informs a more traditional MDP-based algorithm, somewhat similar to the problem presented by Knox and Stone [4]. Though their problem domain extends reward signals learned from human input to a straightforward MDP algorithm like SARSA(λ), the authors' work nonetheless mirrors our problem formulation. While we learn reward signals from labeled data and not human input, we still extend the initial portion of learning to a second portion, in which we optimize a deep Q-network using the initial learned reward signal.

Engel et. al explore the idea of music generation using generative adversarial networks (GANs) [5]. Instead of approaching the music generation problem from an audio waveform approach and using a recurrent model, the authors instead represent the labeled musical data as spectrograms in the frequency domain, then use a convolutional network to learn representations. In addition, instead of relying on explicit music theory rewards to introduce novelty in the generated compositions, the inclusion of a discriminative network as part of the GAN ensures that the generator does not simply replicate or mix the samples from the training data. Finally, instead of generating notes in a step-by-step sequential fashion, the generator outputs an entire composition at once - in this case, if we wanted to find an output sample close to an existing snippet of music, we could simply search the latent input space of the generator.

Another approach to producing natural-sounding drum samples was taken by Gillick et. al [6]. The authors sought to create a sequence2sequence model that takes a MIDI drum beat pattern and outputs a musical performance which not only follows the drum pattern, but incorporates a level of humanization. As opposed to our approach, which incorporates RL techniques with exploration to enforce a "naturalness" prior on generated music, Gillick et al. encode the microtimings and velocities of expert human drummers as matrices and incorporate them into the labeled dataset.

It is also necessary to have diversity in music to make for an interesting track. Chung et. al introduce a model that learns diversity across all samples while maintaining a consistent style across the course of generation [7]. In the case of drum beat generation, this is exactly what we want - enough diversity to maintain interest, but maintaining a style across an entire track. The paper differs from our project in that it applies its model to sequential handwriting generation, which has higher-dimensional structure. However, intuitively, we can expect it to work for time-series generation as well, because it inherently has lower dimensional structure.

In the case of drum beat generation, it is possible to separate each drum into its own track to make polyphonic (multi-channel) music, as opposed to generating drum sounds categorically on a single track. Boulanger-Lewandowski et. al explore a way to model polyphonic music probabilistically by discovering temporal dependencies in high-dimensional sequences [8]. Our project will be multi-channel in the sense of generating independent tracks of drum notes and will mainly require learning of complex rhythms of beats, since melodic and/or harmonic structure isn't as present. The downside of this work in musical application is that longer-term musical structure remains difficult to model. However, in the case of our drum beat generation, longer-term structure is not necessary since our drum beats are only generated for 2 bars, making the paper's findings very relevant to our project.

3 Problem Description

We propose to train a Reinforcement Learning agent to generate future musical samples of a drum beat (i.e. predict what the best next sounds will be) given a starting snippet of a beat. By drum beat, we mean a musical composition consisting of only various percussion instruments.

Our aim is: given a starting portion of a composition, $s \in \mathcal{S}$, and a set of possible percussion instruments to play at the next time step, $\mathcal{A}$, find the policy $\pi^*(a|s)$ that finds the best action $a \in \mathcal{A}$ which maximizes returns with respect to a reward function $r(a, s)$. We expand on the sequential nature of the problem, the state and action space representations we have chosen, and the reward function which informs our agent in the following sections.

3.1 Sequential Structure

The beat generation problem is sequential in the sense that it generates steps of music according to a consistent tempo, where each beat is a time step. At each step, a percussive sound will be generated, based on the sequences of notes that already exist in the composition. A total of 32 beats will be generated to make a sequence.

3.2 State Space

The state of the environment consists of the existing notes placed in the composition. The state is a time series of 32 steps, each representing a 16th note, with each step being a one-hot vector with ten possible categories of instrument to play.

3.2.1 Conversion from MIDI

In order to clearly define the state space, we first converted the MIDI drum dataset into a form that could properly represent all possible combinations of 32 notes in a sequence. Each MIDI track consisted of an arbitrary number of drum notes played over 32 16th beats. The process of extracting and quantizing the values of these drum notes into 16th notes, is shown in Figure 3 and described as follows.

1. The entire track is divided into 32 intervals, where each interval has the duration of a 16th note.
2. Each drum value in the track is grouped into a single interval, such that each interval will contain 1 or more notes. The different notes are shown in the left column in Figure 1.
3. A single note is picked from each of the 32 intervals, which results in a sequence of 32 drum notes. Notes are chosen based on highest velocity value. The velocity value measures the force at which a note is played, so the most forceful (and loudest) note of each interval is chosen.
4. The values of the drum notes are mapped to a smaller space of notes, where similar drum sounds would be mapped to the same value. The mapping is shown in Figure 1, which maps 22 notes down to 9.
5. The resultant sequence of 32 mapped drum notes is converted into a 10x32 matrix with one-hot columns. Each column represents, with a 1, which of the 10 drum notes is chosen at each of the 32 time steps, and 0's otherwise.

3.2.2 The Challenge of State Representation

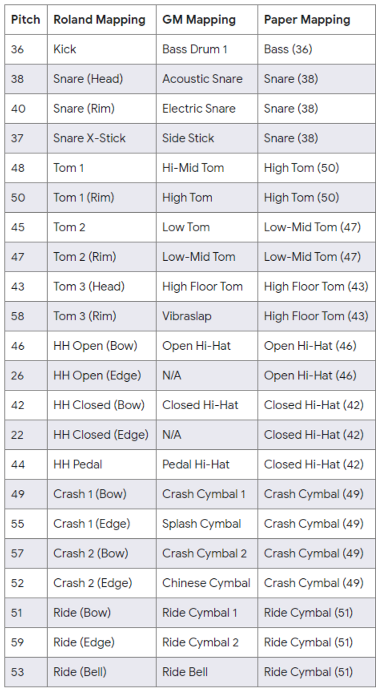
The state space representation includes all of the previously generated drum notes. The challenge is that the number of notes increases with each time step, which increases the size of the state space. Our solution was to zero-pad matrix rows to make all of the states the same size. For example, a state space matrix at time step 3 would include 3 rows of useful information, and the rest of the 29 rows with zeroes. This representation accurately records all previous states and actions of a sequence.

3.3 Action Space

The action space from a particular state consists of all the drum samples that can possibly be played from that state. There are a total of 10 possible actions - 9 drum instruments and a "no event", which is silent.

3.3.1 Action Representation

The action is one time step of a proposed action, and is represented as a one-hot vector of 10 entries, depicting which of the 10 instruments is chosen. The action space will remain static across all states, which means that the same set of possible drum actions are available at every state.



Pitch	Roland Mapping	GM Mapping	Paper Mapping
36	Kick	Bass Drum 1	Bass (36)
38	Snare (Head)	Acoustic Snare	Snare (38)
40	Snare (Rim)	Electric Snare	Snare (38)
37	Snare X-Stick	Side Stick	Snare (38)
48	Tom 1	Hi-Mid Tom	High Tom (50)
50	Tom 1 (Rim)	High Tom	High Tom (50)
45	Tom 2	Low Tom	Low-Mid Tom (47)
47	Tom 2 (Rim)	Low-Mid Tom	Low-Mid Tom (47)
43	Tom 3 (Head)	High Floor Tom	High Floor Tom (43)
58	Tom 3 (Rim)	Vibraslap	High Floor Tom (43)
46	HH Open (Bow)	Open Hi-Hat	Open Hi-Hat (46)
26	HH Open (Edge)	N/A	Open Hi-Hat (46)
42	HH Closed (Bow)	Closed Hi-Hat	Closed Hi-Hat (42)
22	HH Closed (Edge)	N/A	Closed Hi-Hat (42)
44	HH Pedal	Pedal Hi-Hat	Closed Hi-Hat (42)
49	Crash 1 (Bow)	Crash Cymbal 1	Crash Cymbal (49)
55	Crash 1 (Edge)	Splash Cymbal	Crash Cymbal (49)
57	Crash 2 (Bow)	Crash Cymbal 2	Crash Cymbal (49)
52	Crash 2 (Edge)	Chinese Cymbal	Crash Cymbal (49)
51	Ride (Bow)	Ride Cymbal 1	Ride Cymbal (51)
59	Ride (Edge)	Ride Cymbal 2	Ride Cymbal (51)
53	Ride (Bell)	Ride Bell	Ride Cymbal (51)

Figure 1: Drum Instrument Mappings

3.3.2 Dimensionality reduction

The Groove MIDI dataset contains all of the information recorded by the electronic drum set [6]. We made a preprocessing choice to simplify our model. We map all 22 different drums values to a smaller set of 9 drum categories, which consist of the most common instruments in a standard drum kit. Then, we include a "10th" drum value of 0, which represents "no event".

3.3.3 Choice of Monophonic Melodies

A monophonic melody is a sequence of singular notes, as opposed to multiple notes being played at the same time. We chose to generate monophonic melodies, despite the Groove MIDI dataset consisting of non-monophonic melodies. This is because generating combinations of notes scales the action space by a large factor. For example, generating single notes has an action space size of 10, whereas generating pairs of notes has an action space size of 45. Because of this, we chose to maintain simplicity by keeping the action space small.

3.4 Reward Function

The reward function we use is not explicitly coded, and is instead learned through supervised training using expert drum compositions. In order to learn the best short-term actions, we decided to train a neural network to learn the distribution of possible next actions (16th note drum beats) given an existing composition. More precisely, we use inverse reinforcement learning to learn a reward function by training a neural network to predict $p(a|s)$, where $a \in \mathcal{A}$ represents the next drum beat to place in the composition and $s \in \mathcal{S}$ the composition up til that point. We then use $r(a, s) = log(p(a|s))$ as our reward function. Intuitively, this will penalize less likely actions with a heavy negative reward and give only a slightly negative reward for the most likely actions.

We choose to learn the reward function instead of explicitly coding a music theory-based reward for two main reasons. First, we do not have sufficient domain knowledge to properly analyze and implement proper music theory rules for percussion instruments. Second, reward engineering and tuning reward scales is a challenging task which may require as much attention as the learning portion of the problem itself. Since we have access to expert drum compositions in the form of the Groove dataset, we decided it would be optimal to automate the learning of the reward function so as to reduce our potential for error in reward coding.

4 Methods

We base our approach on the work of [1] with some key differences. We use Deep Q-learning, which seeks to learn $Q(s, a, \theta)$ for taking action $a \in \mathcal{A}$ from state $s \in \mathcal{S}$, given the parameters θ in a deep neural network. Then, our optimal policy can be derived from the output of the deep Q-network. Deep Q-learning finds the optimal Q-function by iteratively optimizing the following loss using a stochastic gradient descent-based update:

$$L(\theta) = \mathbb{E}_{\beta}[(r(a, s) + \gamma max_{a'} Q(s', a', \theta^t) - Q(s, a, \theta))^2] \tag{1}$$

where β is the exploratory policy, γ the environmental discount factor, $s' \in \mathcal{S}$ is the state reached from s using exploration policy β, $a' \in \mathcal{A}$ is the action taken from s', and θ^t is the parameters of the target Q-network [1].

In our system, the reward function, target Q-function, and Q-function are each represented by a deep convolutional neural network. Using neural network function approximators allows us to learn policies that are complex and nonlinear in the state and action space when compared to tabular or linear approximation methods [9]. We use convolutional neural networks (CNN) to model the beat sequences instead of the more traditional recurrent networks (RNN) since CNNs have been shown to be capable of modeling time-series data comparably well to RNNs while being less memory-intensive, parallelizing mathematical operations during training and evaluation, and having flexible receptive field size [10].

The first portion of our learning system consists of the supervised training of a CNN to learn the probability distribution $p(a|s)$ of next step drum instrumentation a based on the existing composition s. This Note-CNN serves as the reward function $r(a, s) = log(p(a|s))$. As we explained earlier, learning a reward function using expert trajectories will allow us to ensure we are following good percussion composition rules for musically pleasing beats.

The second stage of the system involves using three neural networks to act as the reward, target Q-network, and Q-network while optimizing a loss function to find the optimal state-action value function. The three networks have the same architecture, with the exception of the final layer of the Q-network and target Q-network having a scalar output ($Q(s, a, \theta)$ and $Q(s', a', \theta^t)$, respectively) and the reward network having an $|\mathcal{A}|$-dimensional output (a probability distribution, $p(a|s)$). The weights of the Note-CNN are used to initialize the reward, Q-, and target Q-networks. By initializing the Q- and target Q-networks with the weights of the Note-CNN, we aim to provide prior information on the values of state-action pairs based on our training data.

During the second stage of our RL system, the reward network is not updated, so it maintains the same function $p(a|s)$ as the Note-CNN. The weights of the Q-network are updated using the loss function (1) with the reward function $r(a, s) = log(p(a|s))$. The weights of the target Q-network, θ^t, are updated using the moving average of the weights of the Q-network, θ. Finally, the optimal policy is found using:

$$\pi(a|s, \theta) = \arg\max_{a \in \mathcal{A}} Q(s, a, \theta) \tag{2}$$

A graphical overview of this system is presented in Figure 2.

4.1 Preprocessing Data

Before training the networks, the percussion data must first be converted into a format that can be accepted into the neural networks. First, we downloaded the Groove-2bar-midi dataset from TensorFlow datasets, which contains 16,389 percussive MIDI tracks, each of which are 32 beats long [6]. Each time step of a track consists of a combination of one or more drums being played in quick succession. To accommodate this to our model, which only accepts 1 beat per time step, we selected only a single sound to represent each time step, based on whichever sound had the highest velocity value. The velocity value indicates the force at which the drum is hit, so at each step, the loudest drum sound was chosen. Next, the sequence of 32 drum notes is converted into a 10x32 one-hot matrix. Each column in the matrix is one-hot, based on whichever of the 10 instruments was played at each of the 32 time steps.

4.2 Training Note CNN

The Note-CNN is a convolutional network which consists of a 1-dimensional convolution layer, a linear layer, and an output layer. The convolution layer extracts the temporal relations within and between instrument channels in time and outputs a high-dimensional feature map which is then flattened into a vector and has its dimensionality reduced in the linear layer before being output. We use ReLU nonlinearities after the convolution and linear layers, and softmax at the output layer to produce a probability distribution $p(a|s)$.

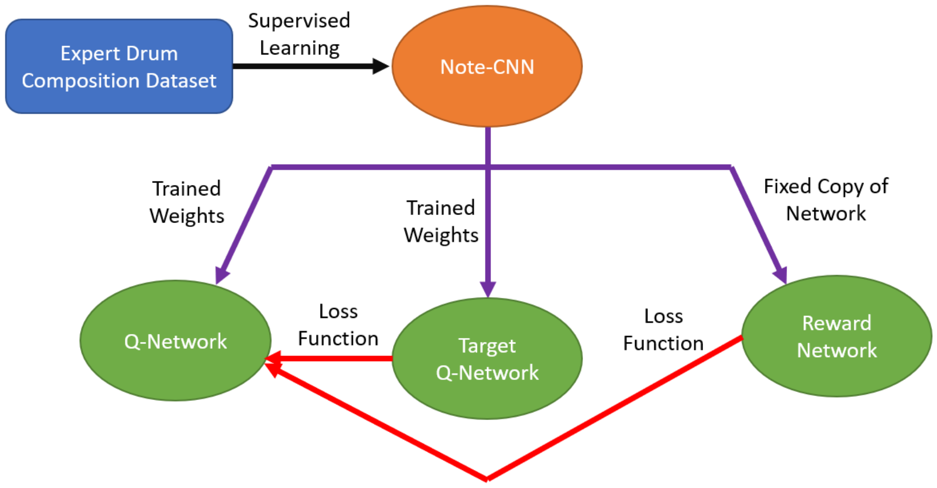


Figure 2: Overview of the reinforcement learning system we employ for this task.

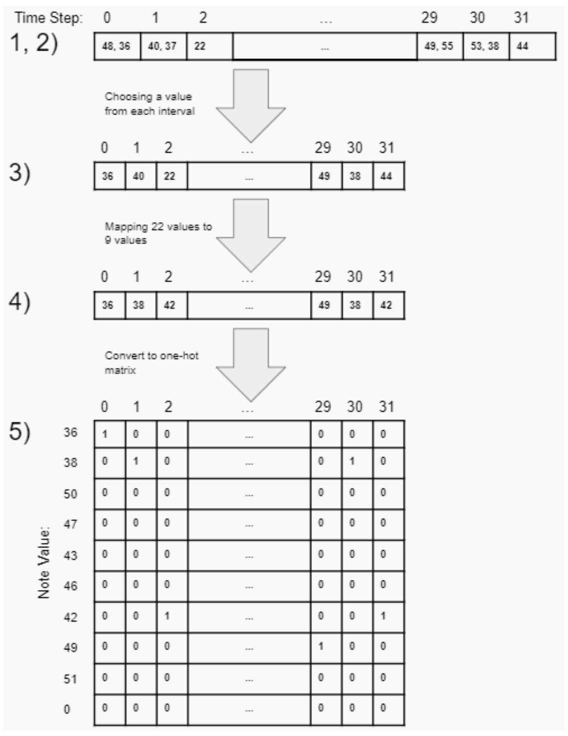

Time Step:
1, 2)
Choosing a value from each interval
3)
Mapping 22 values to 9 values
4)
Convert to one-hot matrix
5)
Note Value:

Figure 3: Conversion from MIDI to one-hot matrix

We train the Note-CNN on the TensorFlow Groove dataset [6] pre-processed into the state representations discussed above. We begin by feeding it one time step of a training sample (zero-padded to make it a valid state representation), calculating the loss between the true instrument played at the next time step and our network's probability distribution and backpropagating it. This is repeated with the next time step, until an entire sample of two measures has been shown to the network, after which we train the model with another sample. Following the reasoning of [1], we do not train the network on the first eight steps of each sample, as we can not reasonably expect the model to predict next actions based on such limited information.

We implement the Note-CNN (and therefore the three networks in the subsequent RL system) on PyTorch [11]. We performed training updates in the manner discussed above, with random sample selection using the Adam optimizer with learning rate 10^{-3}. We chose this learning rate value after performing a coarse search and finding the value with lowest loss on a held-out validation set. We use cross-entropy loss between the network output distribution and the category of drum instrument that is actually played at the next time step. We performed 150,000 iteration steps with random sampling on the Groove MIDI dataset [6].

4.3 Training RL System

The reinforcement learning portion of our solution involves the three networks we introduced earlier: the reward, Q-, and target Q-networks. Each of the networks is initialized with the pre-trained weights of the Note-CNN. The reward network is held fixed throughout the training process. The weights of the target network are updated every 10 iterations based on the rule $\theta^t_{\tau+1} = (1-\eta)\theta^t_\tau + \eta\theta$. We set the update rate $\eta = 10^{-2}$ following [1].

The Q-network is trained by first priming it with a random single time step and following an ϵ-greedy policy for action selection. We tested various values for ϵ and present the results in the following section, but used 0.1 as our default. The action taken by the policy is used to find the next state and reward, then the network weights are updated with the loss function from eq. (1). We train the Q-network for 500,000 iterations and again use Adam optimizer with learning rate 10^{-3}, finding it to be good for the Note-CNN which has the same architecture as the Q-network. We set the discount rate to be $\gamma = 0.5$ following the work done in [1].

5 Results

We analyze the training and performance of our reinforcement learning system. Since the primary purpose of our system is to produce musically pleasing drum compositions, we focus our experiments on human listening tests. Using the same initial weights from a single trained Note-CNN (trained as described in the Methods section), we train several versions of the Q-network used to derive different greedy optimal policies, $\pi(a|s,\theta) = \arg\max_{a\in\mathcal{A}} Q(s,a,\theta)$, which we then compare to each other.

We produce MIDI drum composition samples by selecting a random percussion instrument to play for the first sixteenth note to use as s_0 then using greedy action selection with the selected Q-network to place drums in the composition until the composition has 32 time steps, or two bars of sixteenth notes. We also produce samples from the Note-CNN itself to use as a baseline model without reinforcement learning. The samples for the Note-CNN are produced by drawing a random s_0 as above, then performing stochastic action selection with the output of the network $p(a|s)$ until the composition has 32 time steps.

5.1 Choice of Exploratory Policy in Training

In order to test the effects of exploration on the performance of our model, we train five Deep Q-networks with $\epsilon = 0.01, 0.05, 0.1, 0.3$, and 0.5 for action selection and track a running average of their rewards. Although ϵ-values beyond 0.1 are generally not recommended, we choose to evaluate higher values to determine if there really was a significant effect on the observed rewards. Because rewards indicate only short-term goodness-of-value, we reasoned that more exploratory policies might find a more optimal policy for good-sounding music - especially in our problem setting which is highly dependent on our definitions of state and action space and which can perhaps benefit from some variation in the types of produced beats. In addition, since we are using a log-probability as our reward, the reward variance can be very high and therefore a more exploratory policy might be required to find an optimal policy.

Figure 4 shows the comparison of rewards in a rolling average window of length 1000 between Q-networks trained with $\epsilon = 0.01, 0.05, 0.1, 0.3$, and 0.5. Clearly, the general trend is that the lower the ϵ-value, the higher the average reward. The best average reward is obtained for $\epsilon = 0.01$ and 0.05, with little difference between the two, and the worst reward is obtained for $\epsilon = 0.5$. While this is in line with expected behavior for most RL models, it nonetheless illustrates that a large degree of exploration is not required to optimize our model in this domain, despite the high variance of rewards and the possible positive effects on musicality from a more random policy.

Another important fact illustrated by this comparison is the effect of initialization on the Q-network. Since it is initialized with the trained weights of the Note-CNN, which also acts as the reward function, the Q-network has a strong prior that tends toward the actions with highest probabilities learned during supervised training. Because of this fact, the actions given the highest value by the Q-network for each state already encode the highest-reward outcome at each time step. Therefore, having a less exploratory, more greedy policy will most often select the actions with the highest state-action value and by proxy, the actions which produce the highest reward based on the supervised training.

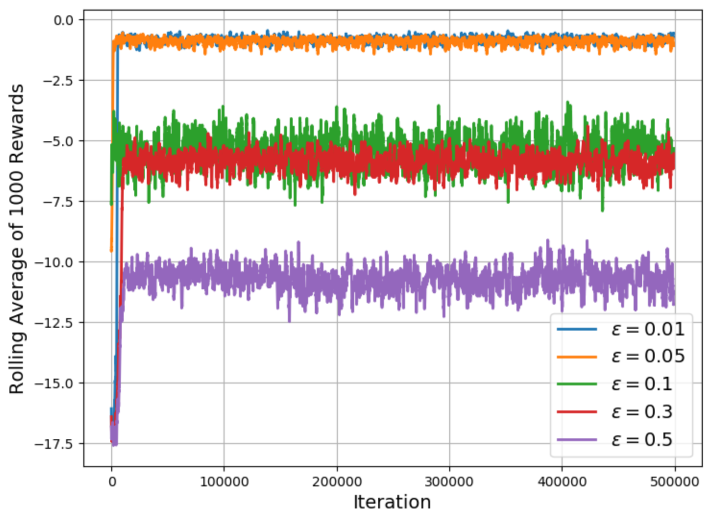


Figure 4: Comparison of rolling average rewards between Q-networks trained with an ϵ-greedy policy with various values of ϵ.

5.2 Performance of RL Models

We compare the performance of the five different policies we learn from each of the Q-networks with different ϵ values. We use the Note-CNN as a baseline with which to compare these methods. We asked 5 participants to participate in a survey in which we played two randomly selected samples and asked the participants to select the sample they preferred, multiple times. We collected 150 total comparisons, where each model was involved in 50 of the comparisons. We chart the categorical preference data in Figure 5.

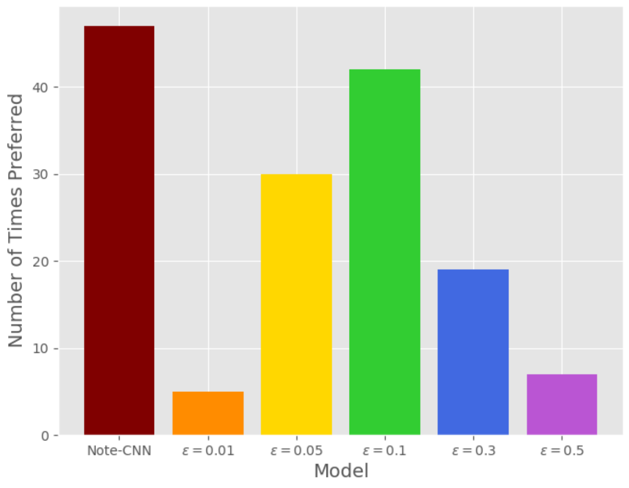


Figure 5: Human preference comparison of drum samples produced by the various music generation models. The sample quality of 5 Q-networks trained with epsilon values $\epsilon = 0.01, 0.05, 0.1, 0.3$, and 0.5 and a baseline Note-CNN model trained on a drum composition dataset are compared in pairwise preference tests with random sampling. The number of times each model is preferred in a comparison is shown.

As clearly shown in Figure 5, there is a surprising amount of preference for the "simple" Note-CNN model which does not incorporate any reinforcement learning. A χ^2 test of significance shows that there is a statistically significant difference in preference between the models, with $\chi^2 = 62.32$ and $p < 0.01$. The second most preferred model is the Q-network with $\epsilon = 0.1$, the middle of the ϵ-values we tested. This is somewhat surprising as well, since the $\epsilon = 0.1$ model did not attain the highest average reward during training. This means that our learned reward function may not have been a good indicator of musicality or the pleasant-ness of a composition. On the other hand, the learned reward function is itself the Note-CNN, which produced the best results in head-to-head comparisons, indicating that supervised training over a drum dataset produced a good generative model for music.

When listening to samples produced by each model, it is clear that the Note-CNN and $\epsilon = 0.1$ models produced the most varied compositions, using multiple instruments in each with tastefully-placed rest notes. On the other hand, the compositions from the other four models tended to repeat one instrument an inordinate amount of times, interspersed with an occasional rest or different instrument. One peculiar behavior is that each of these four models each tended to prefer a different instrument to repeat in their compositions. This may be due to the randomness of initialization during training as well as a preference in the reward function for compositions which repeat one drum instrument. Overall it is clear that our reinforcement learning approach did not fare as well as simple supervised learning, most likely due to an improper representation of the state and action spaces and lack of proper parameter tuning due to constrained computational and time resources. We know that it is possible for the RL models to outperform the supervised model for piano compositions, as in [1], so in future extensions of our work, we aim to achieve this performance for drums.

6 Discussion

The approach of using supervised learning, then transferring that knowledge as a prior for reinforcement learning-based sequence modeling, is a promising way of solving a sequential problem. Supervised learning can help solve the problem of lacking a proper reward function, or having to finely tune a suitable reward function to learn a proper policy. In addition, the weights passed on from the supervised portion of the training to the reinforcement learning portion serve as an additional prior on the initial values of the Q-network.

However, there were many challenges involved with both posing the problem of drum beat generation as a reinforcement learning task and training deep networks to solve the task. We had to make some simplifying decisions during the formulation of our state and action spaces and encountered the problem of having too many variables in designing our network architecture. In potential future extensions of this work, we would like to explore ways of improving the performance of our learning agent by altering some of the choices we made in our solution.

6.1 Challenges with Problem Setup

6.1.1 Challenge of categorical time-series

The challenge of categorical time-series is the case where individual data points in a time-series are not correlated based on distance. Instead, each point is independent. Furthermore, time-series data is usually continuous, like a sequence of amplitudes. However, in the case of percussive instruments generation, the time-series data is a discrete sequence of categories. This requires further data processing to convert it into a format suitable for our design. The solution was to encode the data in a one-hot matrix, which essentially converts each instrument to a separate channel for prediction.

6.1.2 Losing expressivity of beats with quantisation

A large amount of information is lost in the process of converting MIDI files to the one-hot matrix. First, sub-16th-note duration notes lose their precise timings due to notes being grouped into 16th note intervals. More complex beats will have multiple notes within the span of a 16th beat duration, so our method of simplification significantly reduced the quality of the beats. Second, note velocities are overlooked due to the problem design of only generating instrumentation beats, rather than dynamics. Doing so eliminates some of the emotional aspect of the beats since the loudness is not accurately conveyed. On a similar note, the precise timings that contribute to the emotional flow of the music are lost during the conversion because they are separated into evenly distributed intervals.

6.2 Challenges with Neural Networks and RL

The first major challenge we encountered was when trying to extend the code used in [1]. The TensorFlow-based system used for monophonic piano melody generation was geared for piano MIDI generation and did not allow for training of a custom Note-CNN. We therefore chose to re-implement the algorithms from the original paper, using network architectures of our choice. While this allowed us to more finely tune the trade-offs associated with our models and gave us greater control over, for example, the choice between RNN and CNN architecture, it increased the complexity of the problem with regards the number of hyperparameters we would have to set. Given more computational resources and time, we would ideally perform a fine gridsearch over the hyperparameters of both the neural network and reinforcement learning problem formulation. However, within the scope of our project, we instead tuned the hyperparameters of our problem formulation and training process mostly using coarse searches or by following the recommendations from [1].

Another major challenge and limitation we encountered was our lack of an explicit music-theory based reward function. While the trained Note-CNN provides a reward function based on expert human trajectories, it can not learn more complex motifs or rules for musically-pleasing drum compositions. Without a reward that incorporates explicitly-coded music theory, the Q-network will tend toward policies which may simply mimic the playing of the human experts while being less expressive. Future extensions of our work could include a principled, explicit percussion-theory reward.

7 Conclusion

We used supervised learning of expert drum compositions to learn a reward function for deep Q-learning. In addition, by using the trained weights of the supervised model to initialize the Q-network, we imposed a prior on the initial values of the value function that would help guide the reinforcement learning process. We implemented this process on the problem of sequentially generating drum beats to produce a drum composition that would sound pleasing to the ear. We encountered problems during the stages of state and action space definition as well as during training of the deep models.

Our results showed that the supervised model itself produces the most musically-pleasing compositions, followed by a model using a moderately exploratory policy. Although this would seem to indicate that there is no point in tuning a generative sequential model with reinforcement learning, we identified several areas which could potentially have influenced our result and produced a poorer-than-optimal result from deep Q-learning. In future extensions of our work, we seek to find a better state and action space representation as well as to improve the training of our deep models.

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] D. Eck and J. 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, Sep. 2002.

[3] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 1–, New York, NY, USA, 2004. ACM.

[4] W. Bradley Knox and Peter Stone. Combining manual feedback with subsequent mdp reward signals for reinforcement learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 5–12, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[5] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. 2019.

[6] Jon Gillick, Adam Roberts, Jesse Engel, Douglas Eck, and David Bamman. Learning to groove with inverse sequence transformations. In *ICML*, 2019.

[7] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2980–2988, Cambridge, MA, USA, 2015. MIT Press.

[8] Nicolas Boulanger-Lewandowski, Y. Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. volume 2, 06 2012.

[9] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[10] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *ArXiv*, abs/1803.01271, 2018.

[11] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.