Northeastern University
Khoury College of
Computer Sciences

Spotify Sequential Song Recommendation: A Reinforcement Learning Approach

DS 4900 Final Report

Matthew Martin

Khoury College of Computer Sciences, Boston, MA, U.S.A
Submitted 23 April 2021

Abstract

In this paper, we explored the domain of recommendation systems specifically within the area of song recommendation. This application was investigated through implementing a reinforcement learning (RL) algorithm. The basis behind using an RL method is that the nature of the problem is a sequential decision task and using a reward based goal oriented technique is more appropriate than other traditional machine learning methods. The two algorithms implemented were the REINFORCE algorithm and the A2C algorithm. Each was trained using two states one being a single and the other a three song representation. The dataset used is a subset of the Music Streaming Sessions Dataset[1] from Spotify. The dataset consisted of 50,704 songs and 167,880 streaming sessions. The not skip rate of the dataset was 33%. Based on the dataset and training capabilities the models performed adequately. Through experimental results it was found that there are a lot of factors that can contribute to a cap on performance. The best performing model had an average not-skip reward of 5.33 songs for a 20 song session. The code to this paper can be found at the following link.

I. Introduction

Recommendation systems are now in every aspect of our society from clothing, to movies, to music. These systems can help increase user retention and revenue for businesses. This is very important for businesses like Spotify and Netflix where revenue is based off of subscriptions. Optimizing a user's experience will help gain a loyal customer base and increase profits. Recommendation systems have been around for a long time and prior methods include more naive implementations such as content

based and collaborative filtering methods. One limitation of these methods is that they do not account for dependency across timesteps and recommendations.

In 2018, Spotify released the Music Streaming Sessions Dataset (MSSD) as part of a competition on CrowdAI in order to predict sequential skips in a listening session. The complete dataset consists of over 160 million listening sessions and over 3.7 million unique tracks. There is information about user interactions, such as whether or not they skipped a song during a session and metadata features for each track. Each song has 30 tabular metadata features.

The two sets of data are linked by a song's track id. This dataset is suitable for the task at hand due to the fact that it includes skip decisions made by users which makes offline simulation possible. This labeling of skip decisions allows us to focus on the model and algorithm and not on simulation or data collection.

The objective of this project was to explore different RL architectures in order to make meaningful song recommendations. In this paper, we will explore previous works within the song recommender system space and explore implementations of architectures such as REINFORCE and A2C within this domain.

II. Related Work

There are many situations in which RL solutions can be applied. Historically, RL methods have been mostly used in robotics, games, or recommendation systems in which simulation is accessible and easy to do. Within the realm of recommendation systems, specifically song recommendation, there have been many different approaches. Traditional approaches include content based and collaborative filtering methods. Collaborative filtering methods tended to perform better in comparison to content based methods. This stems from research conducted by Slaney realizing that song similarity is inherently unimportant because ratings are an adequate indicator of song similarity[2].

Other research has shown that there can be an upside to using content-based song recommendations. Oord et al.[3] discovered that by creating latent vectors for song representations, via a convolutional neural network, allows for a possible deep learning approach to music recommendation. It has also been found that within a session it is important to note the inner relationships between consecutive songs. Liebman et al.[3] emphasizes this and also introduces a new representation of songs based upon their audio features. In this method they implement a novel RL algorithm that combines song similarity and song transition preferences. This model used online learning simulation to train.

Researchers from University of British Columbia[4] also tackled the song recommendation problem with reinforcement learning. In this study they modeled the problem as a Markov decision process. The work incorporates user preferences of song transitions in addition to song preference. A convolutional neural network was used to model the song and transition features. To reduce exploration time and optimize the model Monte Carlo Tree Search was used.

In terms of more recent and advanced methods there have been attempts at solving this issue with deep reinforcement learning. A project from Harvard University[5] tackled this problem by implementing a Deep Determining Policy Gradient (DDPG) model. Metadata features of songs such as their sound acousticness and bounciness were used to model songs. An autoencoder and long short term memory network were used to compress data features and compress a sequence of songs into a single state. The DDPG model uses an actor critic architecture. Reward was modeled after the not-skip probability and a diversity measure between the current song and the next recommended song. Incorporating deep reinforcement learning showed promising results that could handle large action spaces in a time dependent sequential setting.

III. Experimental Setup

A. Data Preprocessing and Exploration

For the purposes of this project and due to memory and computational restrictions a subset of the MSSD dataset was used. This subset consists of over 50,704 unique songs and 167,880 songs within sessions. There are 10,000 distinct listening sessions. Every song is represented at least once within the sessions data. Each song has 30 metadata features. Examples of these features can be seen in figure 1.

track id	duration	release year
popularity	**acousticness**	**beat strength**
bounciness	danceability	**dyn range mean**
energy	**flatness**	**instrumentalness**
key	**liveness**	loudness
mechanism	mode	**organism**
speechiness	tempo	time signature
valence	**acoustic vector**	

Figure 1: Song metadata features

The dataset has a no-skip rate of about 33%. The length of listening sessions ranges from 10 to 20 songs. A distribution of session lengths can be seen in figure 2.

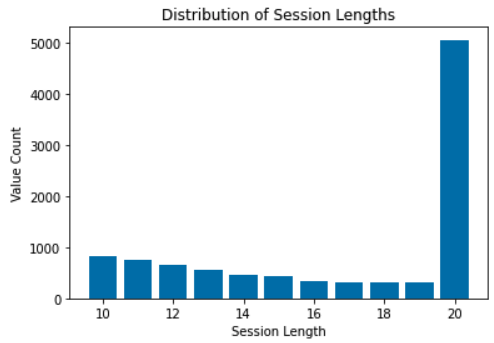

Figure 2: Session Lengths

In order to speed up learning and help with convergence the song features were standardized using the standard scaler method. In addition, three categorical features (key, mode, time signature) were excluded from the dataset due to their low variability within the dataset. For training we needed to obtain the first song from each session and map it to is features to convert it to a tensorized version to create a training set with dimensions of (#songs x #song features). This was accomplished by subsetting the sessions dataset to only include the first song of each session then mapping the track id to the corresponding song metadata and then storing those values in a list to then convert the final list to a torch tensor. In addition, we needed to convert the song feature dataset to a torch tensor.

B. Reinforcement Learning Background

Reinforcement learning (RL) is a subfield of machine learning in which an action is taken given an environment and a reward is given for the action taken. An agent is the model that is trained to select a specific action. In this case the action the agent makes is picking the next song. The environment is the state of the world the agent operates in. The current song being played or the previous n songs played in a session would be the environment. The reward is an evaluation of the action. The reward for a song recommendation system could be for whether or not the user skips the recommended song or not. This can be represented by 1 if the user listens to the recommended song and 0 if the user skips the recommended song. A diagram of this architecture can be seen in figure 3.

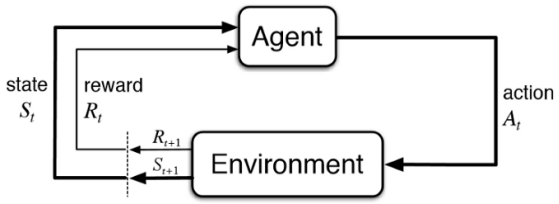

Figure 3: RL Architecture

In RL there are various approaches that can be used. The most common are policy-based and value-based approaches. A policy is the strategy an agent uses to choose the subsequent action based on the current state. A value is the expected long term return with a discount. This value is identified as the long term return given a specific state. Policy-based methods try to find an optimal policy which dictates what action an agent takes and tries to maximize reward. Value-based methods try to optimize a value function and the value obtained from a certain policy.

There is another methodology known as the actor-critic method in which these two methods are combined. In an actor-critic approach the actor decides the best action to take based on an optimal policy, hence the policy-based portion. The critic evaluates this action based off of a value function, hence the value-based portion. These two models work together to optimize one another. This method has proven its ability to learn complex environments with large action spaces. The architecture for the actor-critic method can be seen in figure 4.

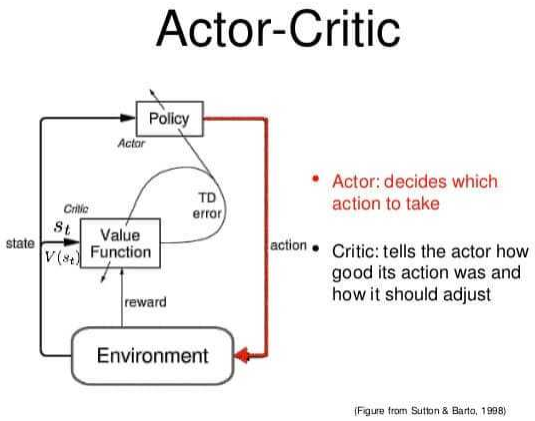

Figure 4: Actor-Critic Architecture

C. Reward Definition and Simulation

In RL methods a reward needs to be defined. Usually, there is an explicit reward that is obtained from an action such as getting a certain amount of points from an action in a game. In online simulations, rewards can also be very explicit based on a user's response such as if a user skips a song or not when it is recommended. A difficulty is determining a reward when an online simulation or a simulator environment is not available. For this project, an offline learning methodology had to occur due to the inaccessibility to users. In addition, the MSSD does not reveal the name of the tracks, only arbitrary track ids. To compensate for these inefficiencies a pseudo simulation was created to generate a reward.

This reward is modeled off of the reward stated in the Harvard project[5]. The reward is a combination of the not-skip probability of a song and a diversity score. The equation for the reward can be seen in figure 5.

$$r = 0.5 * P_{not-skip} + 0.5 * I_{\{dist(s_t, a_t\} > threshold}$$

Where $I_{\{.\}}$ refers to the indicator function. The diversity is measured by the similarity of the current action and previous action.

Figure 5: Reward Function

The not-skip probability is derived by sampling instances from the sessions dataset. Similarity scores are taken between songs by using cosine similarity. Then, the top k similar songs are put into a list. This list is then traversed and each similar song is looked up within the sessions dataset for every instance. For each instance the next song is then compared with the corresponding action song for similarity. If the song is within one standard deviation of the action songs similarity the not-skipped value is then recorded. If a song was not skipped this value is 1 otherwise it is 0. The not-skip probability is a weighted average of these similar songs where the values are whether the song was not skipped and the weights are the respective distances with the current action song. The pseudocode for this method can be seen in figure 6.

The diversity score is determined by the similarity between the current state song and the recommended action song. If the similarity between the two songs is more than two standard deviations away from the state song's similarity with itself then the diversity value is 1 otherwise it is 0. The pseudocode for this calculation can be seen in figure 7.

Current state song: s

Recommended song: a

1. *Calculate cosine similarity between s and all unique songs*
2. *Get top k songs ids with highest similarity*
3. *Calculate cosine similarity between a and all unique songs*
4. *Calculate standard deviation of action similarities*
5. ***For** song in songs id **do** find all instances of song in sessions and get the next song in session: a_session*
    a. ***If** similarity(a, a_session) within 1 std of similarity(a,a) **then** take not_skip value and similarity score*
6. *Not_skip probability: weighted average of not_skip values and similarity score weights*
7. ***Return** not_skip probability*

Figure 6: Similar Songs Pseudocode

Current state song: s

Action song: a

1. *Calculate cosine similarity between s and all unique songs*
2. *Distances: 1 - cosine similarities*
3. *Calculate standard deviation of distances*
4. ***If** dist(s,a) > 0.4 * std **then** diversity score 1 else 0*
5. ***Return** diversity score*

Figure 7: Diversity Score Pseudocode

D. REINFORCE Implementation

First, it was important to begin with a fundamental model in order to understand the workings behind RL algorithms. To do so, it was decided to begin with a policy gradient method. The method selected to implement was the REINFORCE algorithm introduced by Williams from Northeastern University in 1992. The idea behind this algorithm is to reinforce good actions until reaching an optimal policy. Gradient ascent is used to maximize the loss function. This is due to the fact that we are trying to maximize a reward. The algorithm updates the model's weights in a manner that will increase the log probability of selecting a specific action when in a specific state. The log probabilities are optimized instead of the original probabilities due to the gradient of log probabilities being more well-scaled. These log probabilities are then multiplied by the cumulative discounted rewards obtained during each time step. Finally, a gradient of the negative mean of these values is taken to update the model weights. This is the loss value for the model. An equation for the loss function can be found in figure 8.

$$\mathcal{L}(\theta) = -\frac{1}{m}\sum_{t}^{T} ln(G_t \pi(a_t \mid s_t, \theta))$$

Figure 8: Loss function for REINFORCE Model

The algorithm for REINFORCE can be seen in figure 9.

The model used for this method was a feed forward neural network. The network consisted of one hidden layer and used ReLU activation function. The hidden layer is then run through a softmax resulting in the model's output. The output size is the number of unique songs. The model used the Adam optimizer. The model trained for 500 episodes. The loss was updated after each episode and back propagated. There were two different forms of training, one in which the

Input: a differentiable policy parameterization $\pi(a|s,\theta)$
Algorithm parameter: step size $\alpha > 0$)
Initialize the policy parameter θ at random

(1) Use the policy π_θ to collect a trajectory $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, ...a_H, r_{H+1}, s_{H+1})$

(2) Estimate the Return for trajectory τ: $R(\tau) = (G_0, G_1, ..., G_H)$
where G_k is the expected return for transition k:

$$G_k \leftarrow \sum_{t=k+1}^{H+1} \gamma^{t-k-1} R_k$$

(3) Use the trajectory τ to estimate the gradient $\nabla_\theta U(\theta)$

$$\nabla_\theta U(\theta) \leftarrow \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$$

(4) Update the weights θ of the policy

$$\theta \leftarrow \theta + \alpha \nabla_\theta U(\theta)$$

(5) Loop over steps 1-5 until not converged

Figure 9: REINFORCE Algorithm

state was a single song and one where the state was a representation of the previous three songs. This representation is a summation of the features of the three songs.

E. A2C Implementation

To develop a more complex model that would hopefully yield better results an actor-critic approach was implemented. The specific technique picked was the Advantage Actor-Critic (A2C). A2C involves two models: the actor and the critic. These models perform in tandem and use gradient ascent to find a global maximum (reward). A2C improves the actor-critic method by incorporating advantage values. The advantage function determines how much better an action is compared to others at a given state. This function is the temporal difference error between the state action value and the state value. If the advantage value is positive the model will be encouraged to take that action more and if the advantage value is negative it will encourage the model to not take that action. This helps reduce variance and stabilize the policy

network. The equation for the advantage function can be seen in figure 10.

$$A(s_t, a_t) = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)$$

Figure 10: Advantage function Equation

Advantage values allow us to learn how good that action taken is and also how much better it can be. The update equation for this method can be seen in figure 11.

$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$

Figure 11: A2C Update Equation

For the actor model we update the parameters using the policy gradients and advantage value. The loss function is defined as the negative log of the probability distribution scaled by the advantage. This loss is also regularized by an entropy value. This helps improve exploration by discouraging premature convergence of the policy. The entropy value is the negative sum of the mean of the probability distribution from the actor times the log probabilities. This value is then multiplied by an entropy coefficient of 0.0001. This value is added to the actor loss resulting in the final loss. The loss for the critic model is the mean square error of the advantage. The algorithm for the A2C method can be seen in figure 12.

The A2C implementation consists of two neural networks for the actor and critic. The models used for this method were two feed forward neural networks. The actor consisted of one hidden layer and used ReLU activation function. The hidden layer is then run through a softmax resulting in the model's output. The output size is the number of unique songs. The critic consisted of one hidden layer and used ReLU activation function. The output size is the hidden dimension by 1. The model summaries can be seen in figures 14 and 15. Both models used the Adam optimizer. The models trained for 500 episodes. The loss for both was updated after each episode and back propagated. There were two different forms of training, one in which the state was a single song and one where the state was a representation of the previous three songs.

Algorithm 1 Advantage actor-critic - pseudocode

// Assume parameter vectors θ and θ_v
Initialize step counter $t \leftarrow 1$
Initialize episode counter $E \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta)$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t+1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t,\theta_v) & \text{for non-terminal } s_t \text{ //Bootstrap from last state} \end{cases}$
  for $i \in \{t-1,\ldots,t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \nabla_\theta \log \pi(a_i|s_i;\theta)(R - V(s_i;\theta_v)) + \beta_e \partial H(\pi(a_i|s_i;\theta))/\partial\theta$
    Accumulate gradients wrt θ_v: $d\theta_v \leftarrow d\theta_v + \beta_v(R - V(s_i;\theta_v))(\partial V(s_i;\theta_v)/\partial\theta_v)$
  end for
  Perform update of θ using $d\theta$ and of θ_v using $d\theta_v$.
  $E \leftarrow E+1$
until $E > E_{max}$

Figure 12: A2C Algorithm

IV. Results and Discussion

The results for this project were very interesting and took a lot of testing to fully interpret. Initially, the models were trained by updating the loss and backpropagation in batches of episodes but the algorithms call for an update after each episode. This resulted in stagnant losses. Once this was changed to update the loss and back propagate after each episode the loss started to decrease. The issue still persisted that the loss was not decreasing at a meaningful amount. An idea as to why this may be happening is that since the action space is so large the agent gets stuck in the explore phase and does not learn from its previous actions to select new actions based on the policy. In order to increase exploitation, temperature scaling of the logits from the agent output was used. Temperature scaling allows for the transition between a softmax and hardmax. The closer the temperature value goes from 1 to 0 the more the outputs will act like a hard max in which there will be a more clear distribution of action probabilities. This enhances the chance of the agent picking actions that have worked in the past. To implement this the temperature factor starts at 1 and every 30 episodes it is decreased by 0.05 until it reaches a floor of 0.1. Once this was implemented the loss eventually made meaningful decreases and approached zero. A comparison of the losses for the single song state and multi -song state with and without temperature scaling can be seen in figure 13.

The next issue arose when monitoring the loss-reward relationship. Based on how the loss is modeled, being scaled by the reward, as the loss decreases we should see an increase in reward. When training the reward seemed to have a positive relationship with the loss when it should have a negative relationship. This can be seen in figure 14. In order to test why this may be happening the reward in which the model uses for training was changed to solely be the not-skip probability. The reasoning behind this was to see if the diversity part of the reward was overcompensating the reward and not resulting in an appropriate reward. This makes sense due to the fact that the model should begin to pick songs more similar to one another within a session as it learns a user's preferences. Once this change was made the loss-reward relationship became negative. This can be seen in figure 15.

Losses and training rewards for the A2C models can be seen in figure 16. The losses were very sporadic. This may be due to the balancing act of the actor and critic. This leads to a less optimal reward accumulation for listening sessions. Optimizing the models with hyper parameters such as learning rate and using gradient clipping may help stabilize this relationship and yield better rewards.

On the test set we sampled 50 sessions and the results for each of the models can be seen in figure 17. According to these results the best performing model was the sing song state REINFORCEMENT model with an average not-skip reward of 5.33 songs. This was interesting as REINFORCE is the more naive algorithm of the two. I believe this is due to implementation inefficiencies. There was also not a noticeable difference between single and multi state performance.

V. Conclusion and Future Work

From this project, we learned that reinforcement learning can be a very difficult strategy to implement and debug. In addition, reinforcement learning should only be used as a last resort if other supervised or unsupervised methods are appropriate. The simulation aspect is very important and creating a pseudo-simulation resulted in various issues and made it clear that situations in which you can simulate an automatic reward are best suited for this type of solution.

Future work includes trying to test different hyperparameter and reward functions to optimize the current implementations and implementing these methods using test subjects to run this recommendation system as an online learner would be an interesting task to see how performance compares and how the agent learns. Another interesting addition to this project would be to see how Deep Q-learning methods such as DDPG compare to REINFORCE and A2C. An area that would also be interesting to try and discover is the actual songs within the Music Streaming Sessions Dataset to see what these actual listening sessions created by the model are made of and see if the songs are actually similar.

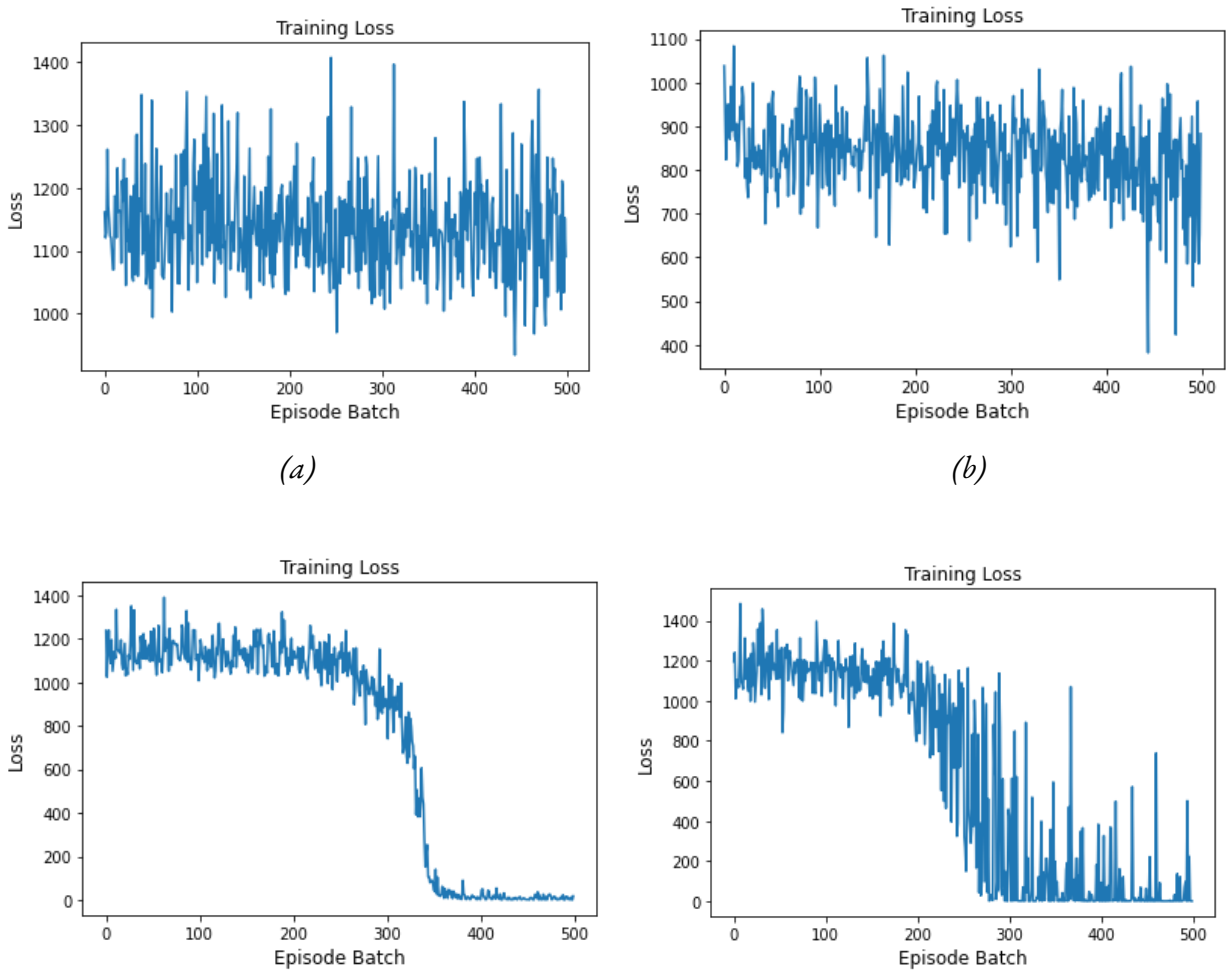

Figure 13: REINFORCE Losses without and with temp scaling. (a) Single Song State without temp , (b) Multi Song State without temp, (c) Single Song State with Temp, (d) Multi Song State with temp

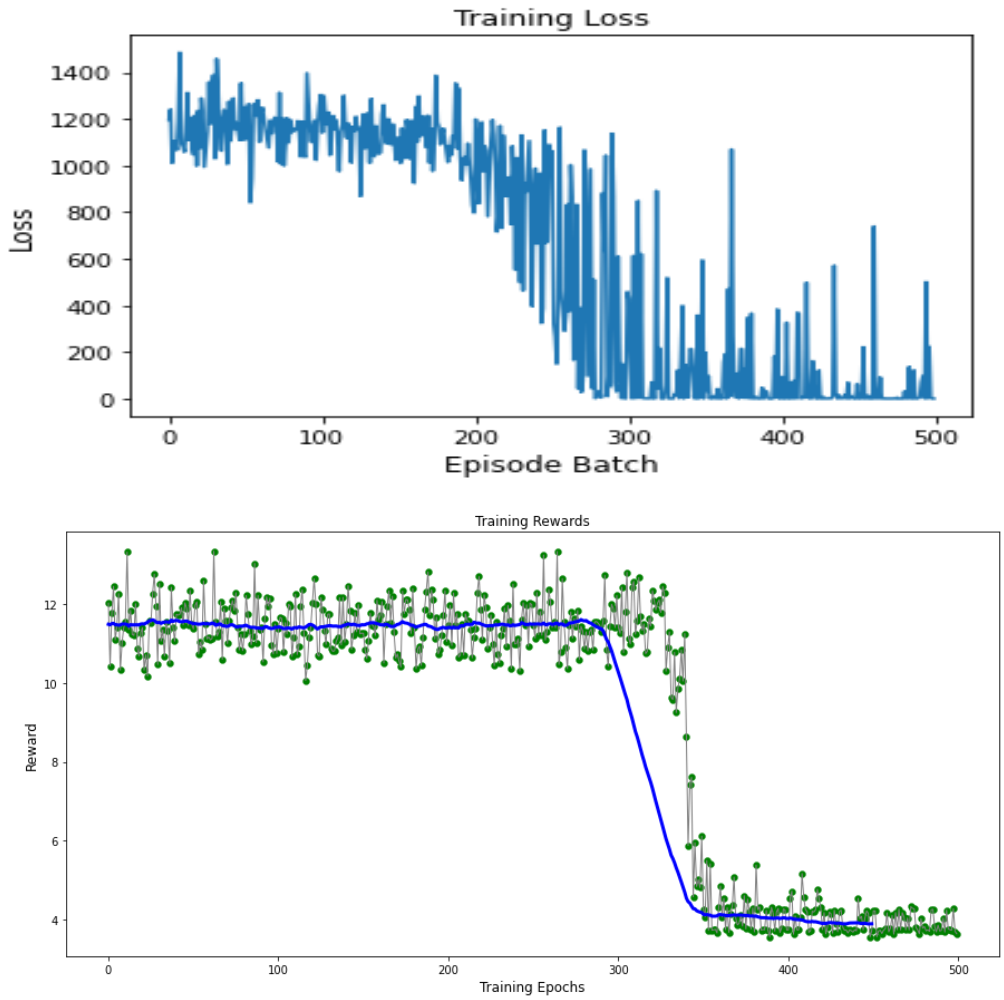

Figure 14: REINFORCE incorrect relationship between loss and reward with original reward function.

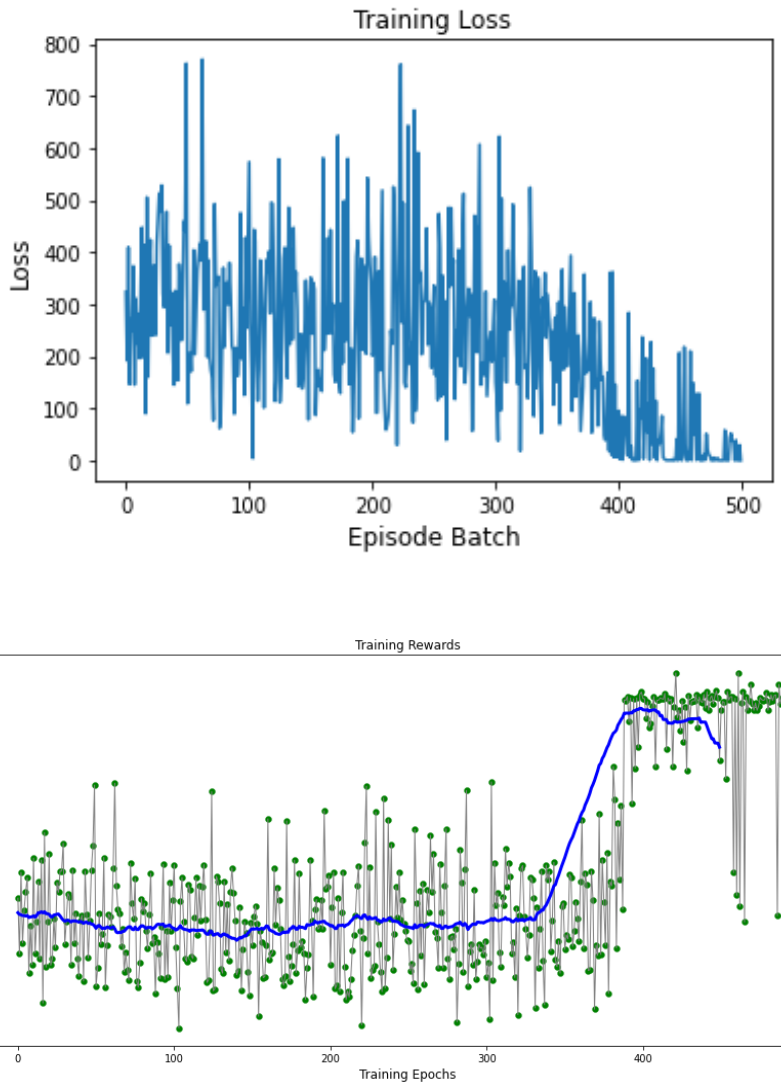

Figure 15: REINFORCE correct relationship between loss and reward with non-skip probability reward function.

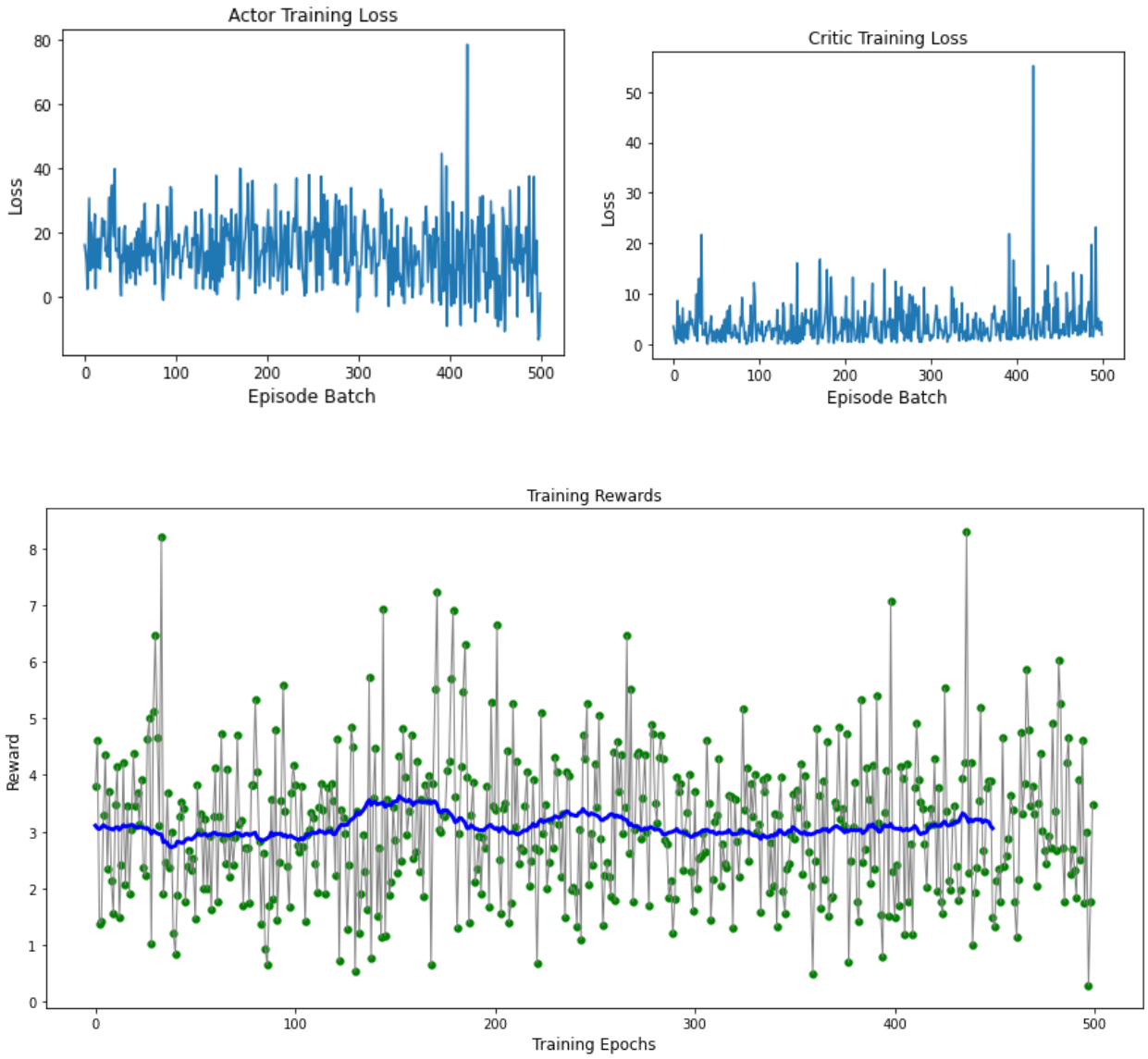

Figure 16: Training Losses and Rewards for Single Song State A2C Model

Model	Average Reward
REINFORCE Single Song	5.33
REINFORCE Multi Song	2.78
A2C Single Song	3.09
A2C Multi Song	2.95

Figure 17: Average Rewards on Test Set

References

1. Brian Brost, Rishabh Mehrotra, and Tristan Jehan. 2019. The Music Streaming Sessions Dataset. In Proceedings of the 2019 Web Conference. ACM.
2. M. Slaney. 2011. Web-scale multimedia analysis: Does content matter? IEEE MultiMedia, 18(2):12–15
3. Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 26, pages 2643–2651. Curran Associates, Inc.
4. Liebman, Elad, et al. 2015. "DJ-MC: A Reinforcement-Learning Agent for Music Playlist Recommendation."
5. https://wendy-xiao.github.io/files/playlist_recommandation.pdf
6. https://sophieyanzhao.github.io/AC297r_2019_SpotifyRL/2019-12-14-Spotify-Reinforcement-Learning-Recommendation-System/