APPLICATION DEVELOPMENT FOR MUSIC RECOMMENDATION SYSTEM USING DEEP DETERMINISTIC POLICY GRADIENT

Rathin S. Kamble, Sujala D. Shetty and Aljo Jose
Department of Computer Science, Birla Institute of Technology & Science, Pilani, Dubai Campus
Academic City, Dubai, U.A.E

ABSTRACT
Recommendation Systems works as an information filtering system which helps to feed and recommend content personalized for the taste of the user. From the use in e-commerce to generic advertisement, recommender systems are proven to be highly effective and go-to solution for personalized content promotion. This project aims to develop and design a Machine Learning model which can be integrated into an Android application to help recommend music for the app user. For this purpose, a Deep Deterministic Policy Gradient model was used along with an underlying architecture for designing the android application which contains playlist of the user’s songs and considering the likes and dislikes from the user, the app with the help of the ML model helps suggest user an additional array of songs.

KEYWORDS
Reinforcement Learning, DDPG, Recommendation System, TensorFlow, BigQuery, Firebase

1. INTRODUCTION
Most of the common recommendation systems model used are designed to maximize the short-term reward while overlooking the fact whether the suggestion would likely lead to a more profitable reward in the long run/long-term reward. This project was implemented using the Keras and TensorFlow libraries to develop and train a Deep Deterministic Policy Gradient (DDPG) model, which takes the recommendation system approach as a sequential interaction between the user and the model, thus leveraging the Reinforcement Learning (RL) methods to automatically learn and update the strategies until the system converges to an optimal policy. Using this approach ensures that the optimal strategy generated maximizes the expected long-term rewards. An android app was developed which holds the user’s music playlist and the trained model was exported to the app using Firebase and BigQuery as a backend service for storing and aggregating the data and mediating the flow of data from the TensorFlow Colab Notebooks and Android application.

2. OVERVIEW OF REINFORCEMENT LEARNING
As the task at hand was for the recommender agent to provide recommendations, we can model this problem with the help of Markov Decision Process (MDP). In MDPs, we have an agent which is interacting with an environment. The agent takes actions, gets next state to proceed to and the reward (scalar value) associated with the action it took. Hence, formally an MDP consists of the tuple of attributes (S, A, P, R, γ).

- State Space (S): Defined as browsing history of the user which are the previous N items that a user browsed before time t.
- Action Space (A): An action At implies a list of items given to user at time t based on the current state St.
- Transition Probability (P): Probability p (St+1|St, At) defines the probability of state transition from St to St+1 when Recommender agent takes an action At
- Reward (R): Based on the current state St and the action taken At, i.e. the agent recommending a list of items to user, the agent receives the immediate reward Rt
- Discount Factor (γ): The immediate reward is considered by a greater factor and the delayed rewards over time are reduced by a factor. This is used to solve the problem of infinite horizon where we must make use of discounted aggregation of rewards achieved over time. (γ ∈ [0, 1])

3. RELATED WORK

This section briefly reviews the related work to this project. In (2016), Paul, C., Jay, A., Emre, S. published paper on how deep learning brought significant improvements to their system. Given that YouTube generates a large amount of data every single hour, they provided various insights on designing and maintaining such massive recommendation system. Rafael Glauber and Angelo Loula (2019) explains the differences and similarities between two main commonly used methods of recommendation systems that is Collaborative Filtering and Content-based Filtering. They proposed to conduct experiments for comparison between the two algorithms for different approaches. Wu et al. (2016) provided a session-based recommendation model using Recurrent Neural Network for real world e-commerce website. They integrated the RNN with Feedforward network which is used to represent the user-items correlation and thus increased accuracy dramatically. Nguyen et al. (2017) implemented a tag recommendation system which makes use of Convolutional Neural Networks (CNNs) for visual information and make use of user’s preferences. Hybrid recommender systems implemented by Robin Burke (2002) are used to enhances the effectiveness of the recommendations ranked by the collaborative filtering method. Shani et al. (2005) implemented an MDP-based recommendation system and made use of an n-gram model which incites on Markov-chain model of user behavior. Peter et al. (2015) implemented a deep reinforcement learning model for Slate MDP for high-dimensional states and action spaces. They introduced a new type of MDP called the Slate MDP and applied deep Q-learning for feature representation of both the state and action space. Mahmood et al. (2009) made use of reinforcement learning strategies for implementing a conversational recommendation agent with the same goal of increasing the cumulative rewards. Taghipour et al. (2007, 2008) looked at the web page recommendation problem as a Q-learning problem and made recommendation based on the web page usage data as the actions and applies the reinforcement learning strategies to constantly learn and attain an optimal policy. X. Zhao, L. Zhang, L. Xia, Z. Ding, D. Yin, and J. Tang (2019) makes use of Deep reinforcement learning for List-wise recommendation strategy to recommend items to user. The model acts on a Markov Decision process and leverages the reinforcement learning methods to come up with a framework to work as an online user-agent interaction simulator giving a list-wise recommendation.

4. METHODOLOGY AND IMPLEMENTATION

In this project, a Deep Deterministic Policy Gradient (DDPG) was used to model the sequential interaction between users and recommendation system. For a recommendation process, given a current state s, the agent recommends a list of items a, to provide the feedback. In our case, the feedback is captured by the user in the app clicking whether the song (item) is liked or disliked. The agent then receives the immediate reward r according to the user’s feedback. Hence to simulate the mentioned process, we need to build a simulator to predict a reward based on the current state and action selected. The data/information from the playlist is processed and used by the model to generate a memory M = {m1, m2, ...} to store users’ historical browsing history where m is a user agent interaction tuple.

The current state and the action recommended by the recommender agent are matched to the historical state action pair such that we can generate a simulated reward. In depth, if the initial state and the historical session are denoted as s0 = {s1, ..., sN} and {a1, ..., aL} then we can observe current state (s), current action (a) and reward list (r) and store the triple ((s,a) → r) in memory. For instance, the recommendation agent recommends a list of five songs denoting as {a1, ..., a5} to the user, then if the user likes a3 and a4, then update the current state by removing the number of items liked from the top of the state resulting in s = {s3, ..., sN, a3, a4}. We calculate the overall reward r of the whole recommended list as follows:
\[ r_t = \sum_{k=1}^{K} I_t^{k-1} u^{k}_x \]  

Here \( K \) is the length of the recommendation list, \( I \in (0, 1] \) and \( u \) denotes the reward permutation list.

For each interaction of training the model, there are two stages. In the first stage, given the current state, the recommendation agent recommends a list of items \( a_t \). The actor (\( f_\theta \pi \)) and critic network (\( Q(s, a | \theta \mu) \)) are initialized with random weight along with the target network \( f' \) and \( Q' \). Similarly, the capacity of the replay memory \( D \) is initialized as well.

The actor then generates a weight vector list using the following equation:

\[ f_\theta \pi : S_t \rightarrow w_t \]  

Here, \( f_\theta \pi \) is a function parameterized by \( \theta^\pi \). Then for each weight the recommendation agent scores the items in the item space. The item with the highest score is selected and added to the recommendation list and the item is removed from the item space to prevent repetitive recommendation.

Then the agent observes the reward from the simulator procedure mentioned above (1) and updates the state accordingly, storing the transitions \( (s_t, a_t, r_t, s_{t+1}) \) into the replay memory \( D \). In the next stage, the recommender agent then samples mini-batch of these transitions from replay memory \( D \) and then updates the parameters of the actor and critic networks based on the standard procedures of DDPG which can be given as follows:-

1. Update Critic:-
   a. Minimizing RPE : \( \delta_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}) | \theta) - Q(s_t, a_t | \theta) \)
   b. Computing the desired output: \( Y_t = r_t + \gamma Q'(s_{t+1}, \pi(s_{t+1}) | \theta') \) and Update \( \theta \) by minimizing the Loss function:
   c. \( L = 1/N * \sum ( Y_t - Q(s_t, a_t | \theta))^2 \)

2. Update Actor using the sampled policy gradient equation.
3. Update Critic and Actor target network:-
   a. Critic = \( \theta^\mu \leftarrow \tau \theta^\mu + (1-\tau) \theta^\mu' \)
   b. Actor = \( \theta^\pi \leftarrow \tau \theta^\pi + (1-\tau) \theta^\pi' \)

5. ANDROID APPLICATION, FIREBASE, AND BIGQUERY

While designing the android app, Android Studio is used which is an integrated development environment (IDE) for Google’s android operating system since it is specifically designed for Android development. Apps can be developed using Java, C++, Kotlin which differs based on programmer’s choice. For this project, Java was used as the preferred language to develop the app. Android Studio has features such as gradle-based build support, android specific refactoring, rich layout editors that allows drag and drop UI components with an option to preview layouts. More importantly, it has a built-in support for Google Cloud Platform, thus enabling integration with Firebase and Google App engine.

5.1 Android App Architecture

Since the project is based on pulling the recommendations provided by the model from TensorFlow which is in a remote data source, a need for using the proper architecture for the android app was essential. For this purpose, the MVVM (Model-View-View Model) architecture is suitable to structure the code such that the code can be more modular and have better separated components such that each part of the program has different responsibility and can be separately modified without affecting other component.

5.2 Firebase

For getting the recommendations, we must gather the data from the app. For this project, making use of Firebase mainly due to Android studio having built-in support for Google Cloud Platform, thus enabling integration with Firebase. Firebase is a useful platform for mobile and web applications. It can be a back end
for many services such as user authentication, data storage, static hosting, real-time database etc. It provides a user-friendly platform for building mobile and web apps. This project enables the Google analytics through Firebase such that with Google Analytics we can obtain the user-behavior data (in this case which song the user likes in a sequential order). This can be done by sending an Analytics event each time the user likes or dislikes the song.

5.3 BigQuery

The Firebase and Google Analytics helps to get the user’s likes and dislikes of song as Analytics events to Firebase. Hence, to store this data and aggregate it, make use of BigQuery. BigQuery is a Google Cloud product that allows you to examine and process large amounts of data. It is serverless, highly scalable multi-cloud data warehouse and it is a great integration with Firebase. The Firebase console for this project was connected with BigQuery so that the analytics data generated by the app is automatically exported to BigQuery into the intraday table and groups events in the events table. The dataset required for Recommendations System needs to be large enough to extract meaningful outputs. So, there is a need to import the sample dataset used into the BigQuery. We used the public One Million Song dataset for this case.

Next step was to create service account credential in the Google Cloud Console to access and load the BigQuery data from the Google Colab environment. This is for importing the BigQuery data, preprocess this data for training the recommendation model and exporting the model in TFLite format suitable for using in mobile app.

5.4 Connection between TensorFlow and Firebase-BigQuery

The analytics data from the BigQuery was imported into the Colab Notebook. For accessing the BigQuery data from the Colab notebook, upload the service account file. Then load the analytics data collected in the app with the Firebase analytics into the notebook using pandas pre-processing libraries to preprocess the data and extract the necessary information needed to feed it as input to the model. BigQuery provides several convenience IPython magics (commands) that will fetch data. Finally, extract the information passed from the app into the Colab notebook.

Once the training of the DDPG model is finished, it was saved and then compressed to export the model into a TFLite file so that it can be used on the App to show the recommendations. The model was then deployed to the Firebase Console using Firebase ML. To do so the use of Firebase Admin SDK was needed.

6. CONCLUSION AND FUTURE SCOPE

The model was downloaded into the app and launched each time you wanted recommendations for the songs you liked. The list of the recommendation can be seen when the floating action button is clicked. The recommendation comes from the model which gives a list of songs from the dataset along with the song liked by the user. Since the model is based on the public One Million song dataset, comparing the mean average accuracy, model is seen to have more diverse selection of music (9.3% vs 7.6%) and better identifies the songs that are skipped by the users (49% vs 54%).

This project was designed to suggest recommendations for music songs while using a reinforcement learning approach. Deep Deterministic Policy Gradient model was developed for this purpose using TensorFlow libraries with the help of Firebase and BigQuery Google Cloud Services to store and aggregate the likes and dislikes of the user through the app. The use of DDPG model acts as a Markov Decision Process and takes advantage of Deep Reinforcement Learning to automatically learn the optimal policy for recommendation of items. Using this ensures that the cumulative rewards over the long run are maximized. There are several research directions for this project, one of which can include applying the model for different applications. Further, addition with other model architectures like RNN, CNN to find better agent-user pattern and investigate how to model these additions mathematically for recommendations would also serve as some future researching prospects to look at.
REFERENCES


