RECOMMENDATION SYSTEMS USING DEEP REINFORCEMENT LEARNING

Nikita Thomas (U20210043), Priyansh Desai (U20210049), Vedika Agarwal (U20210091)
Plaksha University, 2024

Introduction:

In the rapidly expanding landscape of online services, where convenience meets an overwhelming abundance of choices, recommender systems stand as allies, guiding users through the array of available options. These systems are designed to assist users in their quest for tailored recommendations; aligning products, services, or information with individual needs and preferences. The prevalence of recommender systems spans across diverse domains including movies, music, books, points of interest, and social events, reflecting their pivotal role in enhancing user experience and facilitating decision-making processes.

By harnessing the power of DRL, the paper of reference[1] aims to imbue recommender systems with the capacity for dynamic adaptation and long-term strategic planning. Unlike conventional methods, which rely on predefined rules or heuristics, DRL enables systems to learn optimal recommendation policies through interaction with the environment, iteratively refining their strategies over time. It explicitly models user-item interactions for the same by leveraging the capabilities of neural networks to extract complex patterns and representations from data.

Related Work:

The related work in reinforcement learning (RL) for recommendation systems predominantly focuses on model-free RL techniques, as model-based approaches are deemed impractical due to their high time complexity. Among model-free RL techniques, two main categories exist: policy-based and value-based methods.

Policy-based approaches, exemplified by works such as Dulac-Arnold et al.[8], aim to generate deterministic policies mapping states to actions. However, challenges arise, particularly in handling large action spaces and bridging the gap between continuous and discrete action spaces. Similarly, studies like [4], [7] utilize policy networks to output continuous action representations, followed by item ranking based on computed scores. Yet, a common limitation across these works is the lack of emphasis on learning the state representation.
Alternatively, value-based approaches, like those explored by Zhao et al. [5], focus on selecting actions with the highest Q-values. These methods consider user feedback, with techniques such as dueling Q-networks employed to model state-action pairs efficiently. Despite their

effectiveness, value-based approaches face inefficiencies when dealing with a large number of actions.

Problem:

Existing systems treat user preferences as fixed entities, ignoring the inherent dynamics of human taste. This leads to recommendations that become stale over time, failing to capture users' changing interests and needs. This highlights the crucial need to model user preferences as dynamic processes, constantly evolving based on past interactions and current context.

Secondly, many recommender systems prioritize immediate rewards, such as clicks or consumption, as the sole measure of success. This short-sighted approach overlooks the potential long-term impact of recommendations, neglecting items that might not yield immediate engagement but offer valuable benefits in the form of knowledge acquisition, deeper exploration, or even personal growth. While a weather alert might lead to user disengagement, recommending news about a celebrity might trigger further exploration within the same topic, fostering deeper engagement and knowledge acquisition. By solely focusing on immediate rewards, current systems miss out on these valuable opportunities to enrich the user experience and promote long-term platform value.

In this project, we seek to demonstrate the efficacy of DRL by embracing the dynamic nature of online interactions and prioritizing long-term rewards over immediate gains.

Proposed Work:

We propose implementing the architecture proposed in the paper [1] based on the MovieLens dataset for movie predictions using the code repositories provided. This will involve formulating the Recommender and User interactions as a Markov Decision Process followed by the DRR framework based on the Actor-Critic learning scheme. The architecture consists of a deep reinforcement learning based recommendation framework with explicit user-item interactions modeling by feeding the embeddings of users and items from the historical interactions into the multilayer network. The Actor-Critic type framework is incorporated with a state representation module, generating a ranking action to calculate the recommendation scores for ranking to pursue better recommendation performance.

We will then take one of the two following routes:

1. Apply this recommender network to several different domains. Some proposed domains that we will explore are product recommendations for E-commerce sites, music recommendations for music apps and job recommendations.
2. Compare and contrast this DRL based recommender system with non-RL methods to evaluate the pros and cons of each.

Proposed Evaluation:

The most straightforward way to evaluate the RL based models is to conduct online experiments on recommender systems where the recommender directly interacts with users. However, the underlying commercial risk and the costly deployment on the platform make it impractical. Therefore, evaluation can be conducted on public offline datasets using offline evaluation which will utilize the metrics of Precision@k and NDCG@k.

We aim to evaluate our paper based on comparing the results of our model with the original implemented in the paper. We will also evaluate the performance of other recommendation systems not using Reinforcement Learning in comparison to this model.

References:

1. Liu, F., Tang, R., Li, X., Zhang, W. N., Ye, Y., Chen, H., Guo, H., & Zhang, Y. (2018). Deep Reinforcement Learning based Recommendation with Explicit User-Item Interactions Modeling.
2. G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li, "DRN: A deep reinforcement learning framework for news recommendation," in WWW 2018, Lyon, France, April 23-27, 2018, 2018, pp. 167–176.
3. Zhou, X., Zhang, L., Zhia, L., Ding, Z., Yin, D., & Tang, J. (2017). Deep Reinforcement Learning for List-wise Recommendations.
4. X. Zhao, L. Zhang, Z. Ding, D. Yin, Y. Zhao, and J. Tang, "Deep reinforcement learning for list-wise recommendations," CoRR, vol. abs/1801.00209, 2018. [29]
5. X. Zhao, L.
6. Zhang, Z. Ding, L. Xia, J. Tang, and D. Yin, "Recommendations with negative feedback via pairwise deep reinforcement learning," CoRR, vol. abs/1802.06501, 2018.
7. Y. Hu, Q. Da, A. Zeng, Y. Yu, and Y. Xu, "Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application," CoRR, vol. abs/1803.00710, 2018. [33]
8. G. Dulac-Arnold, R. Evans, P. Sunehag, and B. Coppin, "Reinforcement learning in large discrete action spaces," CoRR, vol. abs/1512.07679, 2015.

Timeline:

Our progress was two-fold, gaining a detailed understanding of the DRL architecture used in the paper and implementing the same using the code repositories available to us.

We first went into a deep dive of understanding the nuances of the methodology implemented in the paper. It involved references to various topics explored in class lectures such as MDP formulation, Epsilon greedy methods, Temporal Difference learning and policy iteration. The paper also involved new themes such as priority replay sampling and soft replace strategy along with Deep Learning techniques to create the three main modules used: State Representation, Actor and Critic. This took us around a little over a week.

We then moved on to looking at the MovieLens dataset used and the various features available to us that were used for implementing the model. This involved Exploratory Data Analysis of the dataframes available to us for a few days.

Lastly, the most time consuming part of the project was trying the various code repositories available to us and debugging them to arrive at a working code. This involved trying around 6 code bases, each with their own nuances of trying to understand and implement each of them. We spent a majority of our time here, around two weeks to finally arrive at our current code.

Progress:

1. *Dataset:*

For the project, The MovieLens 1Million dataset from Kaggle was used. The dataset contains 1,000,209 anonymous ratings of approximately 3,900 movies with their respective titles, ids, and genres made by 6,040 MovieLens users with data such as their gender, occupation, age and zip-code. The two are mapped to each other with the ratings information, where each user has rated multiple movies on a scale of 1 to 5. We also store the timestamp of when each of these ratings took place to account for temporal changes in user's preferences. The data is split into training and testing and is converted to embeddings for processing by the rest of the network.\

2. *Code Implementation:*

<u>Training Phase:-</u>

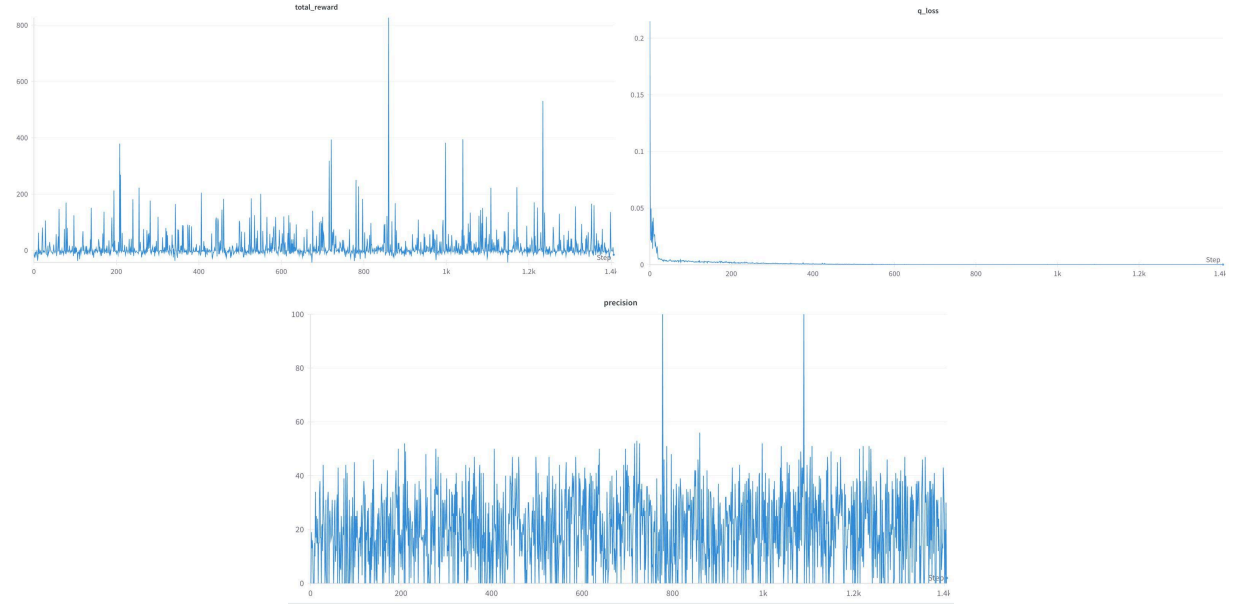
The code runs multiple episodes, for each episode, it resets the environment and while there are still movies to recommend, it trains the actor and critic models, getting the recommended items and the next states, calculating the reward at each step for an episode.

For each recommended item: If the item is in the user's history and has not been recommended before, its reward is calculated based on the user's rating for that item. If the item was not correctly recommended, a default reward of -0.5 is assigned. **If the reward is positive,**

indicating a successful recommendation, the state is updated to replace the oldest item with the recommended one.

The algorithm performs epsilon greedy exploration, with decaying epsilon which gradually reduces the degree of exploration as the agent gains more experience and presumably converges towards better policies. It also computes TD targets by combining immediate rewards with discounted estimates of future rewards, taking into account whether the episode has terminated after taking each action. These targets are then used to train the critic network to estimate the Q-values more accurately. A crucial step in the training process of the actor-critic algorithm, is where experiences stored in the replay buffer are sampled to update the networks, ensuring that the agent learns to approximate optimal policies using the bellman equation.

Some of the metrics we logged while training the network are given below:



Evaluation Phase:-

We trained our model for 1500 epochs and obtained the following precision scores -

```
precision@10 : 0.4397350993377478, ndcg@10 : 0.4255833800930299
```

The precision score is calculated by dividing the total number of correctly recommended items (items with positive reward) by the total number of recommendations (the top-K items recommended). NDCG (Normalized Discounted Cumulative Gain) is a measure of ranking quality that takes both relevance and position of recommended items into account.

A sample episode for Top 3 recommendation:

```
user_id : 4835, rated_items_length:83
items :
 [['Footloose (1984)' 'Drama']
 ['Terminator 2: Judgment Day (1991)' 'Action|Sci-Fi|Thriller']
 ['Toy Story (1995)' "Animation|Children's|Comedy"]
 ['Insider, The (1999)' 'Drama']
 ['Scream (1996)' 'Horror|Thriller']
 ['Silence of the Lambs, The (1991)' 'Drama|Thriller']
 ['Vertigo (1958)' 'Mystery|Thriller']
 ['Rear Window (1954)' 'Mystery|Thriller']
 ['North by Northwest (1959)' 'Drama|Thriller']
 ['Dead Again (1991)' 'Mystery|Romance|Thriller']]
recommended items ids : [1196 2571  260]
recommened items :
 [['Star Wars: Episode V - The Empire Strikes Back (1980)'
  'Action|Adventure|Drama|Sci-Fi|War']
 ['Matrix, The (1999)' 'Action|Sci-Fi|Thriller']
 ['Star Wars: Episode IV - A New Hope (1977)'
  'Action|Adventure|Fantasy|Sci-Fi']]
precision : 1.0, dcg : 2.131, idcg : 2.131, ndcg : 1.000, reward : 2.5

precision : 1.0, ngcg : 1.0, episode_reward : 2.5
```

Challenges and Limitations:

We faced issues on getting the codebase for the paper working. While there were multiple github repositories that implemented the DRL algorithm stated by this paper, many of them contained issues and bugs we were not able to resolve. As a result, we ended up trying 5-6 different repositories to get the final one working. Some of the issues we faced were ‘dimensions based’ and ‘dependencies on installed packages’. The codebases we could explore were also very limited since they had to implement the same architecture as the paper with the same dataset. However, we were able to resolve all the errors for the final repository. One of the limitations we faced was computational abilities. While the paper and other codebases trained the network for 5000 epochs, we were only able to train our network for 1000 epochs which took 3+ hours in itself.

Next Steps:

Following what we have gained so far, we plan to evaluate the performance of 2-3 recommender systems on the MovieLens dataset that do not utilize reinforcement learning to compare and contrast their efficiency. This will help us gain an understanding of the advantages and disadvantages of each and whether RL helps improve the performance of recommender systems overall.

We will also try to implement this DRL model on different domains such as music recommendation, restaurant recommendation and more to see if it performs differently. This will also help us learn if the methodology is to be tweaked due to the different context it is placed in and derive new results for our perusal.

APPENDIX: ***Detailed Methodology***

The paper formulates the problem as an MDP and follows an iterative method similar to that of policy iteration to arrive at the optimal policy and state-action values to update our action to provide the best possible movie recommendations. It is mainly composed of three modules: State Representation, the Actor and the Critic.

MDP Formulation:-

1. **State-** Represents the history of the users **positive** interaction with the recommender. The representation is generated by the State Representation Module by creating a function of the history $[s_t = f(H_t)]$
2. **Action-** It is a continuous parameter vector denoted as $a \in R^{1\times k}$. Each item, $i_t \in R^{1\times k}$ (embedding of item i) has a ranking score, which is defined as the inner product of the action and the item embedding, i.e., $i_t a^T$. Then the 10 top ranked ones are recommended.
3. **Rewards-** The user provides of feedback of click, not click, or rating etc., these are provided as immediate rewards to the recommender (agent) as R(s,a)

Modules:-

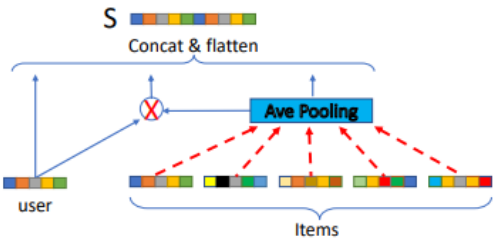


Fig. 6. DRR-ave Structure

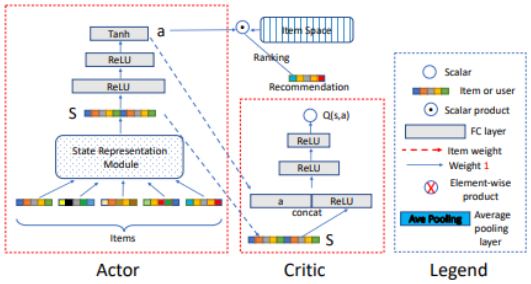


Fig. 3. DRR Framework

1. **State Representation:** To model both interactions of items in the history along with item-user interactions, three components are concatenated and flattened to represent a state in our MDP (as seen in Figure 6).
    a. The embeddings of items in H are first transformed by a weighted average pooling layer.
    b. The resulting vector is leveraged to model the interactions with the input user using an element-wise product operation.
    c. Finally, the embedding of the user, the interaction vector, and the average pooling result of items are concatenate into a vector to denote the state representation
2. **The Actor:** Similar to policy evaluation, this module utilizes two RELU layers and one Tanh layer to generate an action a, based on the current policy π. Using the action vector generated, its product with items are determined to provide scores and the top 10 scored

items are recommended. It utilizes the Epsilon Greedy method to generate the action a. (shown in Figure 3)

3. **The Critic:** It is a Deep-Q network used to approximate the true state-action value function [Q(s,a)]. Based on the values obtained, the Actor module is updated in the direction of improving the performance of action a. The update is based on Temporal Difference Learning using a mini-batch of sequences. (shown in Figure 3)

$$L = \frac{1}{N}\sum_i (y_i - Q_\omega(s_i, a_i))^2$$

$$y_i = r_i + \gamma Q_{\omega'}(s_{i+1}, \pi_{\theta'}(s_{i+1}))$$ Th

Training Algorithm:-

The algorithm uses the interaction history between users and the recommender (agent) as training data. The recommender takes actions following the current recommendation policy $\pi_\theta(s_t)$ upon observing the user's state s_t using ε-greedy exploration and recommends an item i_t based on this action. It then receives feedback (reward) r_t from the user, and the user's state transitions to s_{t+1}. Based on this feedback, the recommender updates its recommendation policy. Finally, the transition (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer D.

In the model updating phase, the recommender samples a minibatch of N transitions using the **prioritized experience replay sampling technique.** This technique employs importance sampling to select transitions efficiently by assigning each transition in the replay buffer a priority value based on its temporal difference (TD) error.

The parameters of the Actor and Critic networks are then updated. Finally, the parameters of the target networks are updated using the **soft replace strategy**, to update target networks gradually rather than abruptly. The parameters of the target network are updated by interpolating or blending them with the parameters of the main network. Instead of directly copying the parameters from the main network to the target network, the soft replace strategy gradually adjusts the target network parameters towards those of the main network. A hyperparameter, τ (tau), is used to control the rate of interpolation.