Proceedings of the International Conference on Industrial Engineering and Operations Management
Dubai, UAE, March 10-12, 2020

Deep Reinforcement Learning Based RecSys Using Distributed Q Table

Ritesh Kumar
Department of Industrial & Systems Engineering
Indian Institute of Technology, Kharagpur
Kharagpur, 721302, India
riteshiitkharag@gmail.com

Ravi Ranjan
Senior Associate Data Science
Publicis Sapient
Bengaluru, 560037, India
ravi.ranjan6@publicissapient.com

Abstract

Recommender Systems are becoming ubiquitous in many settings and take many forms, from product recommendation in e-commerce stores, to query suggestions in search engines, to friend recommendation in social networks. Present systems suffer from two limitations, firstly considering the recommendation as a static procedure and ignoring the dynamic interactive nature between users and the recommender systems. Also, most of the works focus on the immediate feedback of recommended items and neglecting the long-term rewards based on reinforcement learning. In this paper, we propose a recommendation system that uses the Q-learning method [1]. We use ε-greedy policy combined with Q learning, a powerful method of reinforcement learning that handles those issues proficiently and gives the customer more chance to explore new pages or new products that are not popular [4]. Usually while implementing Reinforcement Learning (RL) to real-world problems both the state space and the action space are very vast. Therefore, to address the aforementioned challenges, we propose the multiple\distributed Q table approach which can deal with large state-action space and that aides in actualizing the Q learning algorithm in recommendation and huge state-action space.

Keywords
Reinforcement Learning, Recommender System, Distributed Q Learning, Scalable Application

1. Introduction

Thanks to the explosive growth of information on the World wide web that has provided tons of choices for users. But an overwhelming number of items in the systems also pose a significant challenge for users to get the relevant product they are looking for and that ultimately leads to user unsatisfied. To alleviate such a problem, Recommendation is the most widely used solution. Recommendation systems are majorly classified into two categories: Content-based filtering and Collaborative filtering [2]. In content-based filtering, the recommendation system tries to match the product's features with the user's choice while in the collaborative filtering, the system tries to match user patterns with another user that had the same taste then predict the most user's interest to items. The user's feature is created by the user's historical data [3]. The other one is the Hybrid recommendation system which considers both content-based as well as collaborative filtering approaches. Several studies that empirically compare the performance of the hybrid with the other two methods aforementioned and demonstrated that the hybrid methods provide more accurate recommendations than pure approaches. However, these methods cannot effectively address the following three challenges in recommendation [5]. Firstly, it is assumed that the user's underlying preference is static. However, it is most likely that the user's taste is dynamic in nature. The theory of learning

optimal recommendation policies is evolving quite rapidly. Secondly, the aforementioned studies are trained by maximizing the immediate rewards of recommendations, which merely concentrates on whether the recommended items are clicked or liked, but ignores the long-term impact that the items can make. However, the items with small immediate rewards but large long-term benefits are also crucial. Thirdly is its tendency to keep recommending similar items to users, which might decrease the users' interest.

The reinforcement learning-based recommendation system addresses the aforementioned challenges. We propose a ε-greedy policy combined with Q learning-based recommendation system [7] In this approach, we maintain a Q matrix which maps the best action to take in the given state by using Q value. This framework can consider current rewards and future rewards simultaneously. It works on the feedback data too, and keeps itself updated according to the changing users' taste [9]. But as we can see in the recommendation, both the state space and action space are very vast and maintaining such a huge table is way too difficult to store as well as retrieve.

Therefore, to address the aforementioned challenges, we propose the Multiple\Distributed Q table approach which can deal with large state-action space and that aids in actualizing the Q learning algorithm in the recommendation and huge state-action space.

2. Literature Survey

Many approaches have been made previously to improve the performance of recommendation systems. Different kinds of recommendation techniques like including content-based filtering, matrix factorization-based methods and recently deep learning models. Machine learning binary classification problems, using logistic regression and its variants have also been utilized in recommendation systems. Model-free reinforcement learning techniques have been very recently utilized in recommendations [10]. The relevant literature is currently split into two distinct parts: The reinforcement learning based recommendation system and the other is actualizing Q learning is vast state-action space. Mannor, Shie, et al have used Q learning reinforcement learning approach in recommendation system and further clustering technique to enhance the recommendation [12]. The other reinforcement leaning like Deep Q network has also been deployed in recommendation. In this, Deep Q learning has been used to predict the Q values of the actions in the given state where states are represented as a feature vector. Rojanavasu, Pornthep in his paper "*New Recommendation System using Reinforcement Learning*" used Q learning with local and global table which stored the $Q(s_t, a_t)$ of each customers locally and the global table stored the overall customers' behavior. Our approach of Multiple table was motivated from this local and Global table architecture [6]. The vast state-action space has always been an issue in reinforcement learning. Many approaches like clustering, embedding and others have been used further to deal with it [11]. In "*Deep Reinforcement Learning in Large Discrete Action Spaces"*, G. Dulac-Arnold discussed the architecture in which action embedding is done and then it is mapped to the states using K-nearest neighbors' technique.

Previous works showed that Q learning with the objective to optimize $Q(s_t, a_t)$ can be useful in recommendation method. Regarding vast state-action space, we propose the multiple table architecture to actualize Q learning in recommendations [8].

3. Reinforcement Learning

Reinforcement learning is one of the powerful machine learning algorithms. It is a trial and error scheme where agent tries to learn the appropriate action in the given situation from interacting with the environment and getting the feedback of the action taken.

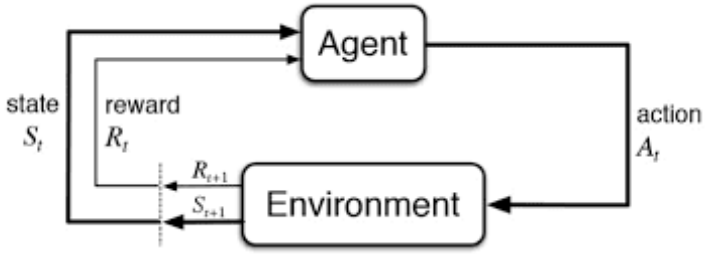

Fig. 1. Reinforcement Learning Mechanism

The main aim of the agent is to maximize the reward which it gets from the user's feedback. As can be seen in the above figure(fig-1) the agent takes an action at a_t state s_t in the environment and shifts to state S_{t+1} after getting reward R_{t+1}. Reinforcement learning is useful when there is no "proper way" to perform a task, yet there are rules the model has to follow to perform its duties correctly. The reinforcement learning can be broadly classified into two classes, the one is Model based and the other is Model free method [1-4]. The model free method can be further classified as Value based and Policy based approach. In this paper, we worked on the value-based approach which is Q learning.

3.1 Q Learning

Q learning is the basic form of reinforcement learning which uses Q values Q(*s,*a) to learn from interaction and iteratively improve the agent's action in the given state. This Q value keeps changing with the users' changing preferences which is captured by the feed-back data. Q(*s,*a) is the estimation of how good is it to take the action *A* in the given state *S* .Q(*s,*a) can be calculated using the formula shown below.

$$Q(s,a) = (1-\alpha)\,Q(s,a) + \alpha\, Q_{obs}(s,a)$$
$$\text{where, } Q_{obs}(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a')$$

3.2 Exploration

The beauty of reinforcement learning lies its exploration technique which differentiates it from machine learning. When the Q(*s,*a) is not optimized, the agent tries to explore it's action space and tries to find the optima action. This avoids the agent to get stuck in the sub optimal value. ε-greedy is the most common exploration technique followed in. The technique is stated as:

$$A = \begin{cases} \text{random(a)}, & \text{if } Q(s,a) < \varepsilon \\ \text{maxaQ(s, a)}, & \text{ifQ}(s,a) > \varepsilon \end{cases} \quad \text{where } 0 \le \varepsilon \le 1$$

4. Methodology and Implementation

We now move on to the method used in this paper. We provide the complete architecture of the recommendation system. For any reinforcement learning based approach we need to build environment according to the formulation of the problem. This environment is problem specific and hence every solution has its own custom environment specific to the problem domain [7]. The OpenAI' gym package provides you access to build custom environment in python. This comes with many inbuilt environment agents like Mountain cars and many Atari games. The environment consists of the functions which deals with action space, shift of states, reset environment and other required functions. Now the algorithm with which we proceeded with is the Q learning with ε-greedy exploration policy as explained in the above section (Reinforcement Learning) In this the system maintains the Q matrix which keeps the changeability of customers' behaviour To update the Q matrix, system gets the reward\penalty from feedback data. It consists of all the possible combinations of state-action pair and the corresponding Q(*s,*a) values

associated with it. This is updated using Bellman equation(eq-1) and the actions are taken using ε greedy policy as stated below in the algorithm section.

	Action 1	Action 2
State 1	0	5
State 2	0	5
State 3	0	5
State 4	20	0

Fig 2. State-Action table

Algorithm

Initialize Q(s,a) arbitrarily or with 0
Repeat for each $s \in S$:
  Choose a∈A in S using the policy derived (ε greedy)
  Repeat(until s terminates):
    Take an action a, observe r,s'
    Choose $a' \in A$ in S using the policy derived (ε greedy)
    Update: $Q(s_t,a_t) \leftarrow Q(s_t,a_t)+\alpha(R_{t+1}+\gamma \max_a \in A Q(s_{t+1},a_t)-Q(s_t,a_t))$
    $s \leftarrow s'$; $a \leftarrow a'$

4.1 Dataset

Before proceeding further with the approach and formulation, we would like to discuss about the dataset which we worked on and implemented our approach. We worked on the movie recommendation system and trained the agent using the very famous MovieLens 100K Dataset.
Total of top 627 movies we picked from the dataset in terms of maximum frequency of the ratings provided to the movie. The movies belonged to multiple genre but for now, we have considered each movie has only one genre.

4.2 Formulation of recommendation in Reinforcement Learning

We now describe the formulation of recommendation system based on reinforcement learning. As reinforcement learning is mostly used in gaming scenario where agents are trained to outperform human players in different challenging scenarios. We model the recommendation system as a game where different actions are taken in different situations to reach the goal. More specifically, the recommendation procedure is formulated as:

Agent- The acting part of the reinforcement learning problem who tries to maximize the reward. It can be said that agent is the model which we try to design. In our model, agent is the recommendation system itself.

State- A state s∈S is the representation of user's positive interaction history with recommender. State is the previous three movies seen by the user st = {movie1, movie2, movie3 } ∈ S

Action- The methods, agent performs to interact with the environment. Every action associated with reward/penalty based on the feedback. Recommendation is the action in our model. at = {a1 , · · · , aN } ∈ A i.e recommending a list of movies.

Reward- After the agent takes an action at∈A in the given state st∈S, it get the feedback from the environment and based on that reward is given to the agent. In recommendation system, the feedback may be click or non-click or implicit.

Environment - It basically consists of the action, state and next state after that action relation. In a state, the agent recommends an item and if that item is clicked then user state changes. That is what the environment consists of state-action-feedback-next_state. For every RL problem to be solved, one needs to design and create the environment, We have designed our own environment for a Recommendation system, how will state change from one state to the other after taking an action. we wrote some functions to make our own environment.
Policy - The policy is the strategy that an agent employs to determine the next action based on the current state. It maps states to actions, the actions that promise the highest reward.

Discount Reward - The discount factor, usually denoted as γ, is a factor multiplying the future expected *reward*, and varies on the range of [0,1]. It controls the importance of the future rewards versus the immediate ones. The lower the discount factor is, the less important future rewards are, and the *Agent* will tend to focus on *actions* which will yield immediate rewards only.

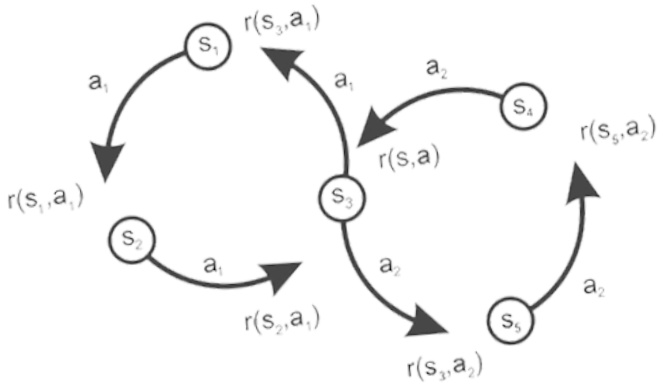

Fig 3. State diagram of changing states

Agent continuously observes the environment and act accordingly. At any instance in time, agent is exactly in one state belonging to potentially infinite state space S where it has to take an action from potentially infinite action space A according to the policy.

4.3 Problems Description

The vast state- action space has always been an issue in the reinforcement learning implementation. Dealing with such an issue has always been a challenging task. From the formulation of recommendation system in reinforcement learning part, it can be seen that in the following we will have as many actions as the number of movies data to recommend which is huge and more dramatic is the state space which is the $^{(\text{number of movie})}P_3$.
Coming to formulation on our dataset, if we keep the previous three movies seen by the users as a state then there will be total of $^{627}P_3$ states which is equal to 245,313,750 and 627 actions. Here we have considered the same three movies having different arrangements as different state. Basically, we are interested in knowing what are the previous movies a user has seen to recommend him/her next movie. So, here order of the movies doesn't matter. So instead of permutations, we dealt with combinations which significantly reduced the number of states to 40,885,625. But still it was a huge number and maintaining such a large Q matrix was way too difficult and taking lots of memories.

4.4 Proposed Solution

In this paper, we propose a distributed table architecture which can be used in implementation of Q learning in vast state-action space. Instead of storing Q(*s,*a) in one table, we made multiple tables which is equal to the number of movie genres in the dataset. Total of 10 genres were there in the 627 movies. We considered only one genre of each movie which made it easier to keep separate table for each genre. Multiple table approach can also be implemented to multiple genre dataset where instead of making separate table of each genre, we need to make tables equal to the combinations of the genres like $^{10}C_3 + {}^{10}C_2 + {}^{10}C_1$, if the movies in the dataset has maximum of three genres. After getting the state of the users to recommend, sub-state is formed which is the states of corresponding table and from each sub-state recommendation is made according to the algorithm explained above and a list of recommendations is provided to the user. As can be seen in the system architecture shown below figure:

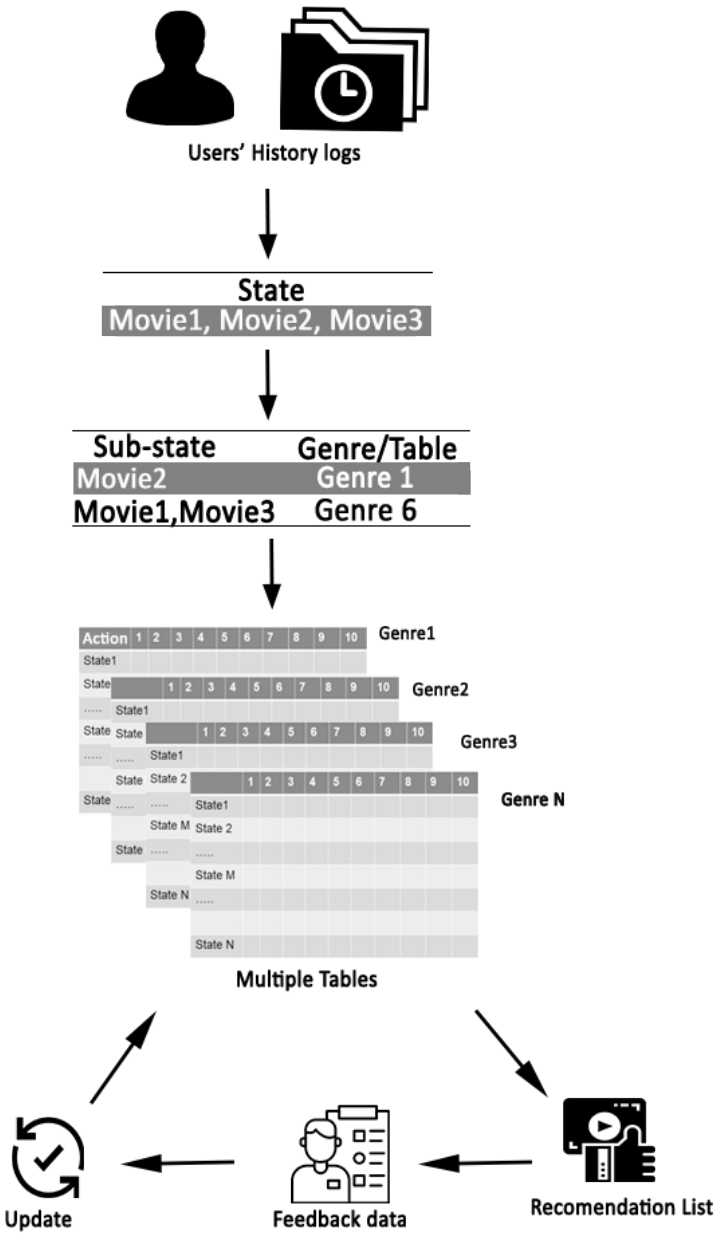

Fig 4. System Architecture

Also, the states in the tables are the combinations of the movies of same genre. We kept the sub-state as ${}^{n}C_1$ and ${}^{n}C_2$. This helped in reducing the states significantly. It may happen that any sub-state had three movies i.e. the sub-state is same as state, in that case we made three sub-states by ${}^{3}C_2$ and then proceeded further same as before. The various other possibilities and which table to access have been shown in the table as reinforcement learning has exploration technique so it's important to note that if we fix the action space to respective genres only, then the agent won't reach the optimal stage and that may lead to irrelevant recommendation so we kept the action space (columns in the tables) same for each sub-state irrespective of its genre. Agent i.e. recommendation system learns from the feedback

data. So, the agent's work doesn't end after providing the list of recommendation, after providing the recommendation list the agent gets the feedback of the user and respective sub-state of the respective genre table is updated using Bellman equation. This is well described in the training section of the paper.

State- Movie1, Movie2, Movie3

Sub-state possibilities	Table
All movies are of different genres *Sub-states*- • Movie1- Genre1 • Movie2- Genre3 • Movie3- Genre5	Look for respective table and then the sub-state in nC1 states of that table
One is of different genre and other two of same henre *Sub-states*- • Movie1- Genre3 • Movie2, Movie3 - Genre5	Look for respective table and then the sub-state in ${}^{n}C_1$ states of that table for sub state containing one movie Look for respective table and then the sub-state in ${}^{n}C_2$ of that table for sub state containing two movies
All three movies are of same genre *Sub-state*- • Movie1, Movie2, Movie3- Genre4	Form ${}^{3}C_2$ sub-states and the look for the sub-states in ${}^{n}C_2$ states of that table

Table 1. State and sub-state scenario table

Recommendation- From the user's history log, we got the state. Now we made a dictionary of sub-state of user's given state $s_t \in S$. That dictionary contained the sub-states, table to which it belonged and row number in that table. Each sub-state (s'_t, s''_t….s'''_t) behaved as state for respective table so we had multiple recommendations in the single state. Sub-state goes in the respective table, looks for the $\max(\max(Q(s'_t,a)),\varepsilon)$ and finally all the recommendations were added to give a list of recommendations to the user. $\text{Max}(Q(s'_t,a))$ is the highest Q value of action a in that state, This is called as greedy action.

5. Training and Evaluation

5.1 Training

In this section, we will discuss how to train Q-Learning based recommender system on multiple Q-table. We will first present the overall idea and then discuss the detailed training process. As aforementioned, Q-Learning utilizes the users' interaction history like click or not-clicked view button as implicit feedback or rating as explicit feedback for the recommended movies. We decided to make multiple Q-tables according to the movie genre. The number of tables is equal to the number of distinct movie genres in the dataset. Every movie is associated with a genre, we took the movies of the same genre and formed its combination in the respective table. But we will be having lots of states if we follow **nC3**, so instead of that, we stored **nC1** and **nC2** of it. This approach will be helpful for scaling the Q learning algorithm. We have considered state as three previous histories, find the genre of past three movies watched using look-up table and formulate the sub-state. These steps are performed while data pre-processing. This will help

in easily accessing all the related information of the sub-state. Now we know the sub-states and its related information, the next step would be to recommend movies according to the sub-state using Q-Learning algorithm. The algorithm will decide whether to explore or exploit according to epsilon value and maximum Q value of that state.

The Q table will be initialized with 0 in the beginning. The algorithm will provide the recommended movie and Q-table will be updated according to user action i.e. click or not-clicked view button as implicit feedback or rating as explicit feedback. This is where Reinforcement learning captures the dynamic behaviour of the user. If the user views or rates the recommended movie then the recommendation was correct so the reward is given to the action in that state and Q value will be updated in the table. If the user ignores to view or rate the recommended movie then the recommendation was incorrect so the penalty is given to the action in that state and Q value will be updated in the table.

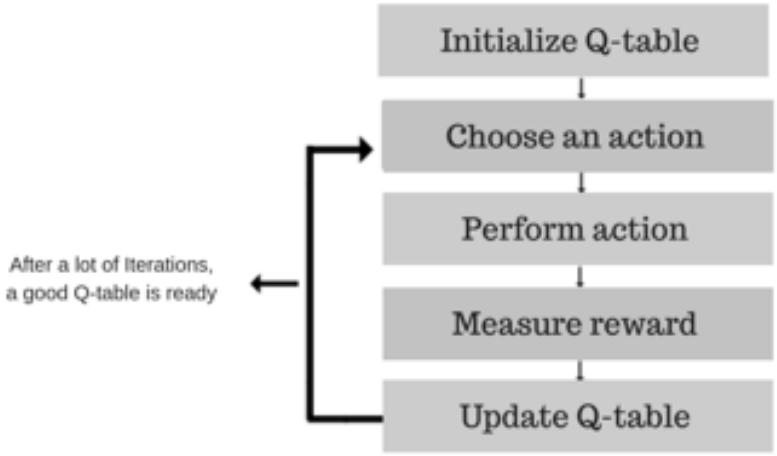

Fig 5. Q-table implementation flow

Feedback data- The very crucial part of reinforcement learning is the feedback data. Getting feedback from the users are way too difficult as users do not want to spend extra time for this. The user feedback is mainly classified into two categories. First the implicit and the other one is explicit feedback. For our paper we have considered the implicit feedback.

Update- Every action of the agent is associated with reward or penalty according to users' feedback data of the action taken. The most important step of the reinforcement learning which helps agent learn the best action to take in the corresponding state and reach optimality. Though the optimal action is not always optimal as the users' preferences keep changing so as the optimal action in the given state that's why a reinforcement learning based model is trained in the entire journey till it is used. As can be seen that in the proposed approach, a single state s_t has many sub-states (s'_t, s''_t..s''' $'_t$) and multiple recommendation has been made. So, for each sub-state's'$_t$, update is to be done. In the previous dictionary we add the recommendation of respective sub-states. Now if the recommendations from the respective sub-state is clicked then that sub-state is updated and it moves to new sub-state $s'_{t+1\_i} \leftarrow s'_{t\_i}$ and then finally all sub states are updated in the same manner and then user comes in a new state $s_{t+1} \leftarrow s_t$. The optimization of Q-table is not possible in most of the cases because of data randomness and there will be very less chances of getting matched with the recommendations. So, if we assign negative rewards to all, then the algorithm will understand that they are not good to recommend but it will not learn which is better to recommend. In order to overcome this, we took the recommendation from Reinforcement Learning and verify it with the actual user action in that state, if it is present there then reward and penalty to others. If it is not in the recommended list then we appended it into the recommended list and assign a reward to it and penalty to others. In this way, the algorithm gets to know which is good to recommend in a particular state along with which is not good to recommend.

5.2 Evaluation

Here in this section, we will discuss how to evaluate the recommender system model performance with various evaluation metrics. The widely used and most common way to evaluate the Reinforcement Learning based recommender system is A/B testing and canary build where the recommender system directly interacts with real-time users. The main disadvantage and risk behind this approach is to set-up the whole infrastructure for model deployment, which is time-consuming and costly. Therefore, we will leverage publicly available offline datasets (Movie Lens) for the evaluation of the proposed Reinforcement Learning based recommender system model. We will perform the evaluation in an offline way on the benchmark dataset comprises of 0.1 million ratings from users to the recommended movies on Movie Lens website.

Offline evaluation: We did offline evaluation on Movie Lens dataset. The offline evaluation of the trained reinforcement learning model is to test the recommendation performance with learned policy using distributed Q-Table. We did offline evaluation on the above mentioned scenario in proposed solution section. The trained reinforcement learning model outperformed with the combination of exploration and exploitation recommendation to overcome the static recommendation.

The problem statement was to make recommendation system using reinforcement learning. We used movie with user up to 150. The movies had multiple genre, we kept single genre of each movie. This data was used to optimize the table. We made state from the data, then predict the recommendation of that state from the RL recommendation and verify which movie was viewed by user in that state. If that movie present in the recommended list then it's good and reward is assigned to it. If not then penalty was assigned.

In most of the cases it didn't happen because the data used to be random and there is very less chance that it matched with the recommendations. So if we assign negative reward to all them, will get to know these are not good to recommend but it won't know which is better to recommend to overcome this, we took the recommendation from RL then verified it with the actual viewed by the user in that state, if it's there then reward and penalty to others. If it's not in recommended list then we appended it into recommended list and gave reward to it and penalty to others. In this way it get to know which is good to recommend in this state along with which is not good to recommend.

Also, in this paper we have provided architecture which can be used to scale Q learning algorithm in vast state-action space. So regarding it's evaluation, we tried to look for it's behaviour whether it's similar ot simple Q learning or not. From the optimized State-Action graph, it can be seen that it is exactly behaving same as simple Q learning but helped in scaling of it. So, it will work same as q learning recommendation system, the add on over it is it can be implemented over vast state-action space and it fully behaves same as simple q learning.

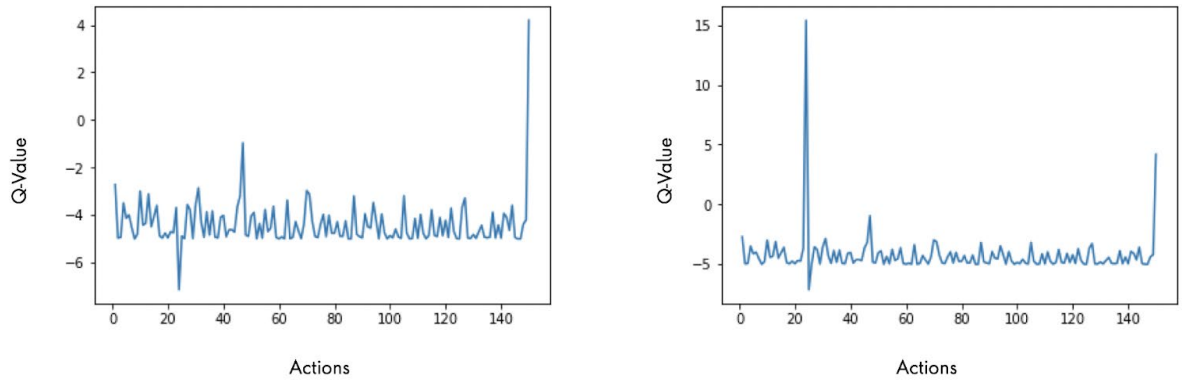

Fig 7. Optimized Q values in a given state and Dynamic nature of reinforcement learning based recommendation system

Reinforcement learning captures the dynamic behaviour of the users from the feed back data. The two graphs shown above(fig-7) are the optimized Q values of different actions at two different times. As can be seen from the first graph, Q value of the last action is maximum, so if any user is in this state then that action will be taken by the user. The second graph is of the same state, as the last action(recommendation) is ignored by the user then because of penalty, the Q value decreased and the Q value of action next to 20 has increased.

5.2 Hyper-parameter tuning

Similar to machine learning algorithms, reinforcement learning too consists of various hyper-parameter . Tuning of these parameters is important for better accuracy and prediction. α, γ, ε, R are some of those hyper-parameter. γ gives the weightage to the future reward, ε let the agent know when to explore. If the value of reward or penalty is too high then there will rapid change is recommendations and if it too low then the motive of capturing dynamic nature may not be fulfilled. So there need to any optimal value of all these parameter which helps in fulfilling our goal of better recommendation. The value of these hyper-parameter also depends on how much traffic you have on your website/app or the others where this recommendation system is used and how frequent you want the agent to change it's behavior because if it is too slow then the recommendation will be irrelevant and which may lead user unsatisfied.

So the tuning of these hyper-parameter is as important as the algorithm itself.

6. Results and Discussion

We captured user engagement with respect to the recommendations provided by the hybrid recommendation system and reinforcement learning based recommendation. We found that user engagement is static when we provide recommendation with hybrid recommendation system where as user engagement increase when we recommend using reinforcement learning models.

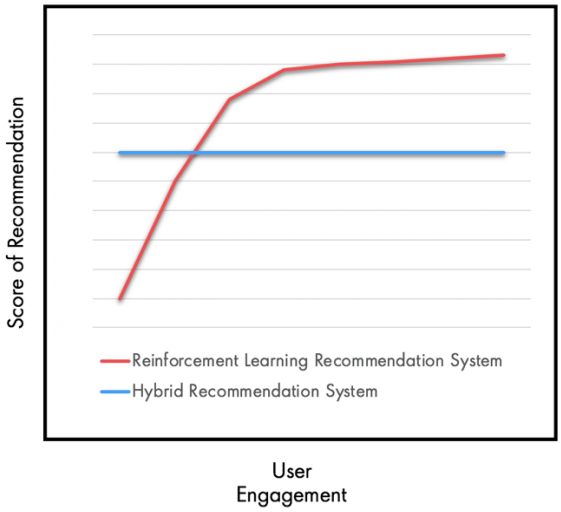

Fig 6. User engagement vs. Score of recommendation

We captured the exploration vs exploitation trend for reinforcement learning based recommendation. We found that after every regular interval, the model provides recommendation using exploration to overcome the static recommendation. This advocates the dynamic nature of reinforcement learning based recommendation system.

References

R. J. Mooney and L. Roy, *Content-based book recommending using learning for text categorization*, in ACM DL, 2000, pp. 195–204

M. Deshpande and G. Karypis, "*Item-based top-N recommendation algorithms*," ACM Trans. Inf. Syst., vol. 22, no. 1, pp. 143–177, 2004. 3 Y. Koren, R. M. Bell, and C. Volinsky, "Matrix factorization techniques

Y. Koren, R. M. Bell, and C. Volinsky, "*Matrix factorization techniques for recommender systems," IEEE Computer*, vol. 42, no. 8, pp. 30–37, 2009

J. Wang, A. P. De Vries, and M. J. Reinders, "*Unifying user-based and item-based collaborative filtering approaches by similarity fusion*," in SIGIR. ACM, 2006, pp. 501–508

W. Zhang, T. Du, and J. Wang, "*Deep learning over multi-field categorical data - - A case study on user response prediction*," in ECIR 2016, Padua, Italy, March 20-23, 2016. Proceedings, 2016, pp. 45–57

Y. Qu, H. Cai, K. Ren, W. Zhang, Y. Yu, Y. Wen, and J. Wang, "*Productbased neural networks for user response prediction*," in ICDM 2016, December 12-15, 2016, Barcelona, Spain, 2016, pp. 1149–1154

Mannor, Shie, et al. "*Dynamic abstraction in reinforcement learning via clustering." Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004.

Liu, Feng, et al. "*Deep reinforcement learning based recommendation with explicit user-item interactions modeling*." arXiv preprint arXiv:1810.12027 (2018).

Rojanavasu, Pornthep, Phaitoon Srinil, and Ouen Pinngern. "*New recommendation system using reinforcement learning*." Special Issue of the Intl. J. Computer, the Internet and Management 13.SP 3 (2005).

Dulac-Arnold, Gabriel, et al. "*Deep reinforcement learning in large discrete action spaces*." arXiv preprint arXiv:1512.07679 (2015).

H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica, "*Ad click prediction: a view from the trenches*," in KDD 2013, Chicago, IL, USA, August 11-14, 2013, 2013, pp. 1222–1230

G. Dulac-Arnold, R. Evans, P. Sunehag, and B. Coppin, "*Reinforcement learning in large discrete action spaces*," CoRR, vol. abs/1512.07679, 2015

Biographies

Ritesh Kumar is a final year undergraduate student pursuing dual degree (Btech+ Mtech) at IIT Kharagpur enrolled in Manufacturing Systems and Master of Science in Industrial Engineering in the Department of Industrial and Systems engineering at the Indian Institute of Technology, Kharagpur.

Ravi Ranjan is working as Senior Data Scientist at Publicis Sapient, Bengaluru. He is part of the Centre of Excellence and responsible for building machine learning model at scale. He has worked on multiple engagements with clients mainly from Automobile, Banking, Retail and Insurance industry across geographies.
In the current role, he is working on a Hyper-personalized recommendation system for the Automobile industry focused on Machine Learning, Deep learning, Realtime data processing on large scale data using MLflow and Kubeflow. He holds a bachelor's degree in Computer Science with a proficiency course in Reinforcement Learning from IISc, Bangalore.