Project Report

Recommendation System using Reinforcement Learning with Generative Adversarial Training

Aathira Manoj (am10245)

Sayali Shukla (sds735)

Abstract

Recommendation systems are widely used in online platforms. Reinforcement Learning is well suited for recommendation systems as they allow the recommendation engine to learn policies that maximize user's overall long-term satisfaction without sacrificing the recommendation's short term utility. Current solutions mostly focus on model-free approaches, which require frequent interactions with the real environment, and thus are expensive. We use a model-based reinforcement learning solution called IRecGAN [14] which models user-agent interaction for offline policy learning via a generative adversarial network. To reduce bias in the learned model and policy, we use a discriminator to evaluate the quality of generated data and scale the generated rewards.

1. Introduction

Recommendation systems have become a crucial part of almost all online service platforms. They are used in a variety of domains to connect users with their most interested items. A common way of building recommendation systems is to estimate a model which minimizes the discrepancy between the model prediction and the immediate user response according to some loss function.

Users' interest can be long-term or short-term. A typical interaction between the system and its users is — users are recommended a page of items and they provide feedback, and then the system recommends a new page of items. Short-term interest reflects the user's immediate interests during the interaction, while long-term interest usually happens after several clicks. For example, clicks are generally considered as short-term feedback which reflects users' immediate interests during the interaction, while purchase reveals users' long-term interests which usually happen after several clicks. This allows us to model the recommendation system as a Reinforcement Learning (RL) agent which aims to maximize user's overall long-term satisfaction without sacrificing the recommendation's short-term utility.

Classical model-free RL methods require collecting large quantities of data by interacting with the environment. This is impractical in the recommendation system setting because bad recommendations (e.g., for exploration) hurt user satisfaction and increase the risk of user drop out. Hence, it is preferable for a recommender to learn a policy by fully utilizing the logged data that is acquired from other policies instead of direct interactions with users.

In this paper, we use a model-based learning approach to estimate the user behavior model from the offline data and use it to interact with our learning agent to obtain an improved policy simultaneously. In spite of being sample efficient, model-based RL suffer from inherent bias in its model approximation of the real environment. To address this, we incoperate adversarial training into the recommender's policy learning. In particular, a discriminator is trained to differentiate simulated interaction trajectories from real ones so as to debias the user behavior model and improve policy learning. This enables robust recommendation policy learning.

2. Related Work

The majority of current solutions are model-free and therefore do not directly model agent-user interactions. Value-based techniques, such as deep Q-learning [1], have unique advantages in these methods, such as smooth off-policy learning, but they are vulnerable to instability with function approximation [2]. On the other hand, policy-based approaches such as policy gradient [3] remain stable but suffer from data bias without real-time interactive control. When importance sampling [4] is used to counter bias, it also results in a lot of variance.

Model-based RL algorithms use a model of the system to predict rewards for state-action pairs that aren't visible. It has been successfully applied to power robotic systems both in simulation and in the real world [5]. It is considered to outperform model-free solutions in terms of sample complexity [6]. However, most powerful model-based algorithms have relied on relatively simple function approximations, which are difficult to implement in high-dimensional space with nonlinear dynamics, resulting in significant approximation bias.

The problems of off-policy learning [7] and off-policy evaluation are quite pervasive and challenging because as the policy evolves, so does the distribution under which the expectation of gradient is computed. In case of recommender systems, item catalogues and user behavior change rapidly requiring substantial policy changes. Therefore it is not feasible to take the classic approaches to constrain the policy updates before new data is collected under an updated policy. Multiple off-policy estimators leveraging inverse-propensity scores, capped inverse-propensity scores and various variance control measures have been developed [8, 9, 10] for this purpose.

For the sequence generation problem, SeqGAN [11] is proposed as a way to extend GANs with an RL-like generator. The discriminator provides the incentive signal at the end of each episode using a Monte Carlo sampling approach. The generator learns the strategy by performing sequential actions and estimating cumulative rewards.

In our model, the generator consists of two components, the recommendation agent (A) and the user agent (U). The interactive process is modeled through adversarial training and policy gradient. We rely on a policy gradient based RL approach called REINFORCE [12], but we simultaneously estimate a user behavior model to provide a reliable environment estimate so as to update our agent on policy. And unlike the sequence generation task, we leverage adversarial training to help reduce bias in the user model and further reduce the variance in training our agent. The agent learns from both the interactions with the user behavior model and those stored in the logged offline data.

3. Method

3.1 Problem Statement

Problem:

Our problem is to learn a policy from offline data such that when deployed online, it maximizes cumulative rewards collected from interactions with users. We address this problem with a model-based reinforcement learning solution, which explicitly models users' behavior patterns from data. A recommender is modeled as a learning agent to generate actions under a policy. Each action generates k recommended items. It interacts with the environment (the user) to produce a set of n sequences, with each sequence consisting of the agent actions (k recommended items), the associated user behavior corresponding to the agent actions (click on a recommended item) and the corresponding rewards (make a purchase).

Assumptions:

We make the following assumptions: 1) Each time, the user must select one item from the list of suggestions. 2) Only clicked objects are eligible for rewards. 3) Items in the recommendation list that are not clicked will have no effect on the user's future behavior.

3.2 Approach

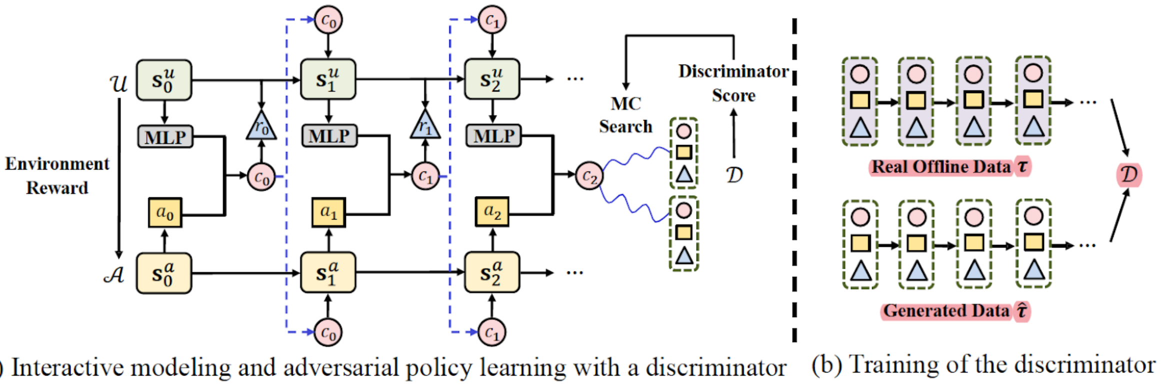


Figure 1: Overview of the interaction between the agent, user and the discriminator

Our model consists of three components: the Agent (A), the user behavior model (U) and the discriminator (D). The figure shows the interaction between these components. The learning begins with the agent (A). The agent generates k recommendations (actions), initially from a distribution and later based on the clicks generated by the user. User (U) generates clicks based on the recommended items. The rewards are generated based on the clicks. Discriminator (D) is trained to differentiate the real data (offline) from the data generated by the model to train the system to capture the intrinsic patterns in the real distribution.

User Behavior Model:
Given users' clicks, the user behavior model first projects the clicked item into an embedding vector e^u. The state $s^u{}_t$ can be represented as a summary of click history. A recurrent neural network is used to model the state transition P on the user side given by,

$$\mathbf{s}_t^u = h^u(\mathbf{s}_{t-1}^u, \mathbf{e}_{t-1}^u)$$

Given the top-k recommendations at time t, the probability of click among the recommended items is computed via a softmax function,

$$\mathbf{V}^c = (\mathbf{W}^c \mathbf{s}_t^u + \mathbf{b}^c)^\top \mathbf{E}_t^u, \;\; p(c_t|\mathbf{s}_t^u, a_t) = \exp(\mathbf{V}_i^c) / \sum_{j=1}^{|a_t|} \exp(\mathbf{V}_j^c)$$

where V^c is the evaluated quality of each recommended item, E^u is the embedding matrix of recommended items, W^c is the click weight matrix, and b^c is the corresponding bias term.
Under the assumption that target rewards only relate to clicked items, the reward r_t for $(s_t{}^u, a_t)$ is calculated by:

$$r_t(\mathbf{s}_t^u, a_t) = f_r\big((\mathbf{W}^r \mathbf{s}_t^u + \mathbf{b}^r)^\top \mathbf{e}_t^u\big),$$

f_r is the reward mapping function and can be set according to the reward definition in specific recommender systems.
User behavior model U can be estimated from the offline data by combining the above equations and using maximum likelihood estimation as follow:

$$\max \sum_{\tau_i \in \Omega} \sum_{t=0}^{T_i} \log p(c_t^i|\mathbf{s}_t^{u_i}, a_t^i) + \lambda_p \log p(r_t^i|\mathbf{s}_t^{u_i}, c_t^i)$$

Agent:
The agent should take actions based on the environment's provided states. However, in practice, users' states are not observable in a recommender system. As a result, we build a different state model on the agent side to learn its states. Similar to the user side, the states are summarized by using a different recurrent network denoted by h^a given by,

$$\mathbf{s}_t^a = h^a(\mathbf{s}_{t-1}^a, \mathbf{e}_{t-1}^a),$$

Based on the current state $s_t{}^a$, the agent generates a size-k recommendation list out of the entire set of items. The probability of item i to be included under the policy is given by:

$$\pi(i \in a_t|\mathbf{s}_t^a) = \frac{\exp(\mathbf{W}_i^a \mathbf{s}_t^a + \mathbf{b}_i^a)}{\sum_{j=1}^{|C|} \exp(\mathbf{W}_j^a \mathbf{s}_t^a + \mathbf{b}_j^a)},$$

where $W_i{}^a$ is the i-th row of the action weight matrix W^a, C is the entire set of recommendation candidates, and $b_i{}^a$ is the corresponding bias term.

Adversarial Training:
We leverage adversarial training to encourage our model to generate high-quality sequences that capture intrinsic patterns in the real data distribution. A discriminator D is used to evaluate

the probability that a given sequence is generated from the real recommendation environment. The discriminator can be estimated by minimizing the objective function:

$$-\mathbb{E}_{\tau\sim data}\log\big(\mathcal{D}(\tau)\big)-\mathbb{E}_{\tau\sim g}\log\big(1-\mathcal{D}(\tau)\big).$$

To deal with partially generated sequences, discriminator uses Monte-Carlo tree search algorithm with the roll-out policy to evaluate to calculate sequence generation score at each time given by,

$$q_{\mathcal{D}}(\tau_{0:t})=\begin{cases}\frac{1}{N}\sum_{n=1}^{N}\mathcal{D}(\tau_{0:T}^{n}),\tau_{0:T}^{n}\in MC^{\mathcal{U},\mathcal{A}}(\tau_{0:t};N) & t<T\\ \mathcal{D}(\tau_{0:T}) & t=T\end{cases}$$

where $MC^{U,A}$ is the set of N sequences sampled from the interaction between U and A.

Policy Learning - REINFORCE:
We use a model based algorithm called REINFORCE which uses Monte-Carlo sampling for Policy Leaning. It works because the expectation of the sampled gradients is equal to the actual gradient. The gradients for the user and the agent are estimated by using the following equations.

Gradient Estimation for User Behavior Model is given by,

$$\mathbb{E}_{\tau\sim\{g,data\}}\Big[\sum_{t=0}^{T}q_{\mathcal{D}}(\tau_{0:t})\nabla_{\Theta_u}\big(\log p_{\Theta_u}(c_t|\mathbf{s}_t^u,a_t)+\lambda_p\log p_{\Theta_u}(r_t|\mathbf{s}_t^u,c_t)\big)\Big],$$

Gradient Estimation for the Agent is given by,

$$\mathbb{E}_{\tau\sim\{g,data\}}\big[\sum_{t=0}^{T}R_t^a\nabla_{\Theta_a}\log\pi_{\Theta_a}(c_t\in a_t|\mathbf{s}_t^a)\big],\quad R_t^a=\sum_{t'=t}^{T}\gamma^{t'-t}q_{\mathcal{D}}(\tau_{0:t})(1+\lambda_r r_t)$$

where, q_D is the sequence generation score, θ_u denotes the parameters of User (U) and θ_a denotes the parameters of the Agent (A). γ is the discount factor and λ_r is the weight for cumulative target rewards. The parts inside the gradients are the same that we had seen earlier during user behavior and agent modeling.

3.3 Implementation

```
Algorithm: IRecGAN
Input: Offline data; an agent model A; a user behavior model U; a discriminator D
  1. Initialize an empty simulated sequences set $B^s$ and a real sequences set $B^r$
  2. Initialize U, A and D with random parameters
  3. Pretrain U and A via the policy gradient equations using only the offline data.
  4. A and U simulate m sequences and add them to $B^s$. Add m trajectories to real data set $B^r$
  5. Pre-train D using $B^r$ and $B^s$
  6. for e = 1 to epoch do
  7.     for r to steps do
  8.         Empty $B^s$ and then generate m simulated sequences and add to $B^s$
```

```
9.          Compute $q_D$ at each step $t$
10.         Extract floor($\frac{\lambda_1}{\lambda_2}$ $m$) sequences into $B^r$
11.         Update U and A according to policy gradient
12.     end
13.     for $d$ to steps do
14.         Empty $B^s$ and then generate $m$ simulated sequences by current U, A and add to $B^s$
15.         Empty $B^r$ and add $m$ sequences from the offline data.
16.         Update D for $i$ epochs using $B^s$ and $B^r$
17.     end
18. end
```

4. Experiments

4.1 Hardware and Software Setup

All our experiments were conducted on GPUs NVIDIA V100 and NVIDIA P100 through Google Cloud Platform (GCP) and Google Colab Pro respectively. The frameworks used for our implementation are Python 3.5 , Numpy 1.18.0 , Tensorflow 1.15.0 and Pytorch 1.1.0. We used Mac OS (version 10.14.6) with Anaconda package management tool for testing.

4.2 Performance Evaluation

4.2.1 Datasets

Our experiments were conducted in both online and offline environments. Subject to the difficulty of deploying a recommender system with real users for online evaluation, we used simulation-based studies to first investigate the effectiveness of our approach following [13].

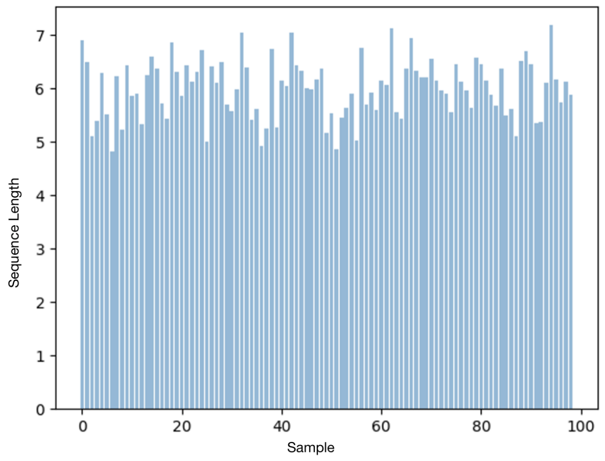


Fig 2: Sample distribution of dataset produced in Simulated Environment 2

Online Evaluation: Simulated Environments

Simulation Environment 1:
Similar to [14], we synthesize a simple MDP to simulate an online recommendation environment. The MDP is modeled using 10 state and 50 items for recommendations with a randomly initialized transition probability matrix $P(s \in S| s_i \in S, a_j \in A)$. Under each state s_i, an item a_j 's reward $r(a_j \in A| s_i \in S)$ is uniformly sampled from the range of 0 to 1. During the interaction, given a recommendation list including k items selected from the whole item set by an agent, the simulator first samples an item proportional to its ground-truth reward under the current state s_i as the click candidate. With the sampled item a_j, a Bernoulli experiment is performed on this item with $r(a_j)$ as the success probability; then the simulator moves to the next state according to the state transition probability $p(s| s_i , a_j)$. The maximum length of each session is chosen as 3 and the average session lengths are 2.81/2.80/2.77 respectively. We initially generate 10,000 sessions with 8000/1000/1000 sessions for training/validation/testing.

Simulation Environment 2:
In order to make our simulation closer to the actual online user interaction with a recommendation system, we model a second MDP. Unlike the first simulation, we do not explicitly specify the number of states and the transition probabilities. Instead, we use an LSTM similar to the Agent and Environment to generate the clicks given the recommendation list. We use 20 recommendation items with a maximum session length of 5. Each session is prefixed with a start symbol <START> and suffixed with an end symbol <END>, resulting in a total length of 7. Fig 2 shows the distribution of the dataset produced. We initially generate a total of 10,000 sessions with 8000/1000/1000 sessions for training/validation/testing.

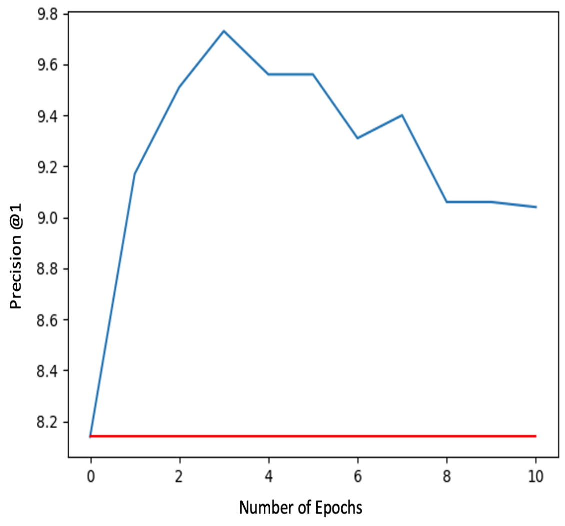
Logging Policy:
Both the simulations generate offline recommendation logs denoted by d_{off} with the simulator. The bias and variance in d_{off} are controlled by changing the logging policy and the size. We use Π_{max}, which recommends the top k items with the highest ground-truth reward under the current simulator state at each step.

Offline Evaluation: CIKM 2016 Dataset
CIKM 2016 Dataset is used for offline evaluation of our model. It is a Personalized E-Commerce Search Challenge dataset containing user sessions extracted from e-commerce search engine logs. Sessions of length 1 or longer than 40 and items that have never been clicked are filtered out. We selected the top 40,000 most popular items into the recommendation candidate set, and randomly selected 65,284/1,718/1,720 sessions for training/validation/testing.

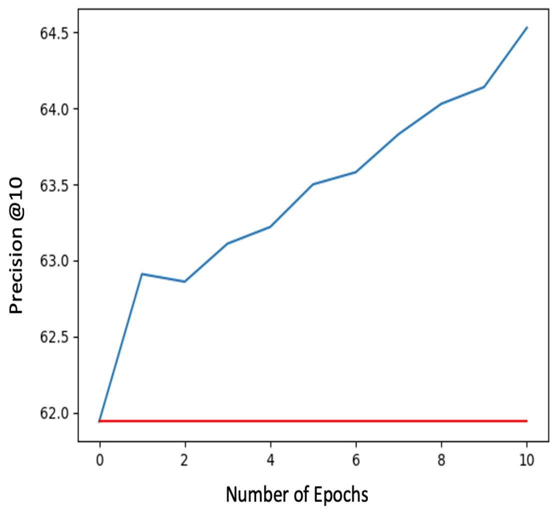
4.2.2 Model Hyperparameters

We use 2-layer LSTM units with 512-dimension hidden states. The item embedding dimension is set to 50. The discount factor γ in value calculation is set to 0.9, the scale factors λ_r and λ_p are set to 3 and 1 respectively. We use an RNN-based discriminator D. The ratio of generated training samples and offline data, λ_1 and λ_2 respectively, for each training epoch is set to 1:10.

4.3 Results



(a) Precision@1



(b) Precision@10

Fig 3: Online learning results from Simulation Environment 2

We have used Precison@k as our evaluation metric. Precision@k is the proportion of recommended items in the top-k set that result in clicks by the user. For example, if precision@10 in the top-10 recommendation setting is 80%, it implies that 80% of the recommendation made by the agent results in clicks/purchases by the user.
We trained our model on both the simulation datasets as well as the offline CIKM dataset for 10 epochs to recommend top-1 and top-10 items. Fig 3 (a) and (b) shows precision@1 and precision@10 on the test set as the training progresses in Simulation Environment 2.

4.3.1 Evaluation

Model	**Baseline LSTM**	**Our Model (IRecGAN)**		
	CIKM	**CIKM**	**Simulation 1**	**Simulation 2**
Precision@10(%)	32.89	**34.06**	**36.5**	**64.06**
Precision@1(%)	8.20	7.23	5.3	8.06

Table 1: Evaluation results on offline and online datasets

LSTM trained without the discriminator and trained using only the offline dataset is taken as the baseline. Table 1 shows the precision@1 and precision@10 on offline and online datasets after training for 10 epochs.

The following observations can be made based on the results:

1) Our model produces encouraging precision@10 improvements against all baselines (+2%). However, it falters with precision@1. LSTM based model produces better precision@1 results. The reason for this could be the explorations for future rewards done by the RL algorithm. When asked to recommend a single item (p@1), the best possible strategy is not to explore and consider future rewards. Instead, it is to recommend the item which produces the best short term reward. That is exactly what is done by the LSTM model. Hence, the LSTM model performs better than the RL based model for precision@1.
2) The model is not able to produce good online results with simulation 1. The results are similar to the offline results. This suggests the inability of the simulator model to mimic the online environment and hence the difficulty of recognizing high reward items with the simulator. It produces much better online results when evaluated with the second simulation model.
3) As expected, precision@10 results are much higher than precision@1 results. It is because the RL agent is able to add items in the recommendation set that maximizes both the long term as well as the short term rewards when asked to recommend 10 items. This strategy fails when asked to predict just a single item.

5. Conclusion

In this paper, we developed a practical solution for utilizing offline data to build a model-based reinforcement learning solution for recommendation. We introduced adversarial training for joint user behavior model learning and policy update. Our model produces encouraging results with precision@10, indicating that it is able to optimize exploration and exploitation better than regular LSTM models. Our evaluations in both simulated and real-world recommendation datasets verify the effectiveness of our solution.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[2] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In Advances in neural information processing systems, pages 1057–1063, 2000

[3] Reinforcement Learning. An introduction, richard s. sutton and andrew g. barto, 1998.

[4] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In Advances in Neural Information Processing Systems, pages 1054–1062, 2016.

[5] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In Proceedings of the 28th International Conference on machine learning (ICML-11), pages 465–472, 2011.

[6] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. Foundations and Trends R in Robotics, 2(1–2):1–142, 2013.

[7] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In Advances in Neural Information Processing Systems, pages 1054–1062, 2016.

[8] Adith Swaminathan and Thorsten Joachims. Batch learning from logged bandit feedback through counterfactual risk minimization. Journal of Machine Learning Research, 16(1): 1731–1755, 2015.

[9] Adith Swaminathan and Thorsten Joachims. The self-normalized estimator for counterfactual learning. In advances in neural information processing systems, pages 3231–3239, 2015.

[10] Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcemen learning. In International Conference on Machine Learning, pages 2139–2148, 2016.

[11] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[12] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3-4):229–256, 1992.

[13] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. Generative adversarial user model for reinforcement learning based recommendation system. In Proceedings of the 36th International Conference on Machine Learning, volume 97, pages 1052–1061, 2019.

[14] Xueying Baiz, Jian Guanx and Hongning Wangy. Model-Based Reinforcement Learning with Adversarial Training for Online Recommendation.