Recommendation System using Reinforcement Learning with Generative Adversarial Training

CSCI-GA-3033
Introduction to Deep Learning Systems

Aathira Manoj (am10245)
Sayali Shukla (sds735)

Executive summary

Problem : The recommender is modeled as a learning agent to generate actions (*k* recommended items) under a policy. The interaction between the agent and the environment (user) produces a set of *n* sequences where each sequence contains agent actions, associated user behavior and the reward. A policy is learnt to maximize the expected cumulative reward.

Solution: We model a Markov Decision Process to simulate interactive user behavior. We adopt different policies to simulate different user behaviors and evaluate the performance of our model on each of the policies and also on different numbers of recommended items (k).

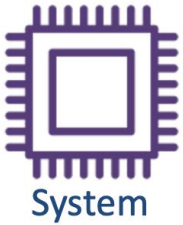
Benefits: We receive a lot of information every day on different platforms. When recommender system efficient and intelligent many industries can generate a significant amount of revenue by using them. Also, it is a way to stand out significantly from competitors.

Problem motivation - Recommender System

- Intelligent system that assists user's information seeking tasks
- Goal: Suggesting items that best match user's preferences

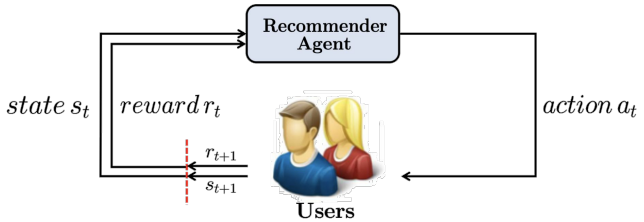
Traditional Recommendation System and limitations

- Collaborative filtering methods
- Content based methods
- Limitation
  - Large dataset required
  - Less intelligent
  - Less adaptable and flexibility
  - Overlooking the long-term influence on user experience



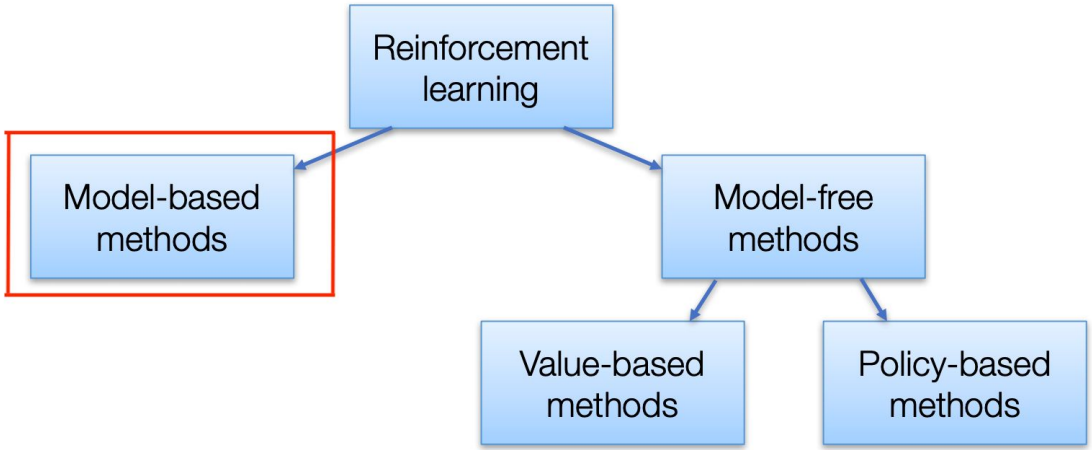
Why use RL for Recommender Systems

- Goal: selecting actions to maximize future reward
- Continuously updating the recommendation strategies during the interactions



- Maximizing the long-term reward from users

Background work



Drawback of Model free methods

- Instability
- Collecting large quantities of data is extremely expensive

Model based Approach

- Model-based RL algorithms incorporate a model of the environment to predict rewards for unseen state-action pairs.
- Model-based RL has a strong advantage of being sample efficient and helping reduce noise in offline

RL with adversarial training

- Dramatic changes in subsequent policy updates impose the risk of decreased user satisfaction, i.e., inconsistent recommendations across model updates.
- To address these issues, we introduce adversarial training into a recommender's policy learning from offline data. The discriminator is trained to differentiate simulated interaction trajectories from real ones so as to debias the user behavior model and improve policy learning.

Related papers and github repositories

Paper 1: Xueying Bai, Jian Guan, Hongning Wang, Model-Based Reinforcement Learning with Adversarial Training for Online Recommendation.

Paper 2: Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. Generative adversarial user model for reinforcement learning based recommendation system.

Referenced GitHub Repository: https://github.com/JianGuanTHU/IRecGAN

Technical challenges

- Estimating User Model and Reward Function with limited data
- Parameter tuning of GAN
- High Computational Complexity
- Balancing exploration with exploitation

Approach

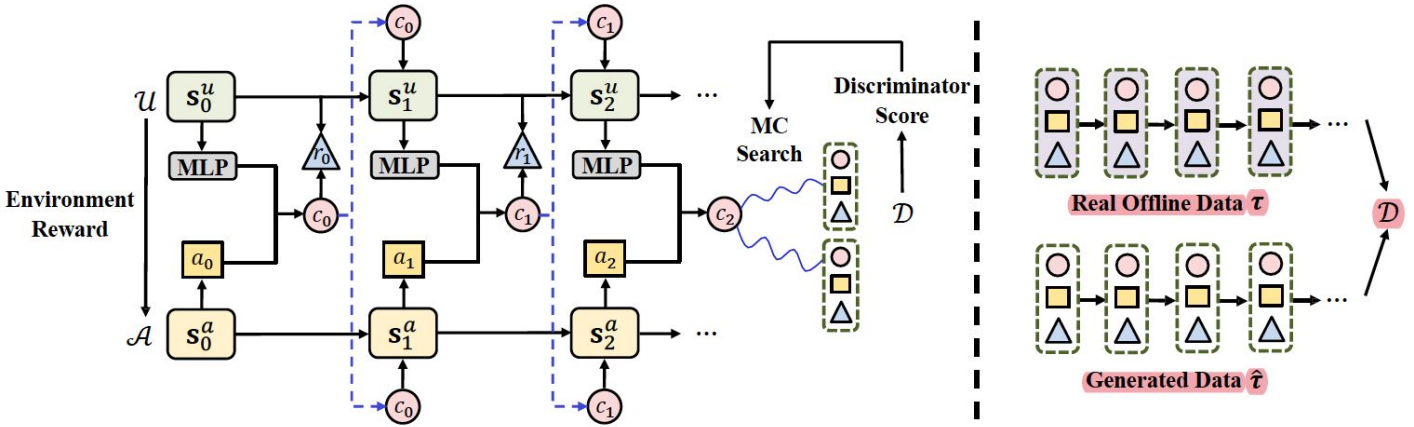
Our recommendation policy is learnt from both offline data and data sampled from the learnt user behavior model, i.e., a model-based RL solution. We incorporate adversarial training in our model-based policy learning to:

- Improve the user model to ensure the sampled data is close to true data distribution
- Utilize the discriminator to scale rewards from generated sequences to further reduce bias in value estimation

Our framework consists of 3 components:

1. User Behavior Model (U)
2. Agent (A)
3. Discriminator (D)

Solution Diagram



(a) Interactive modeling and adversarial policy learning with a discriminator (b) Training of the discriminator

User Behavior Model

Given users' click observations, a recurrent neural network is used to model the state transition on the user side,

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

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

$$\mathbf{V}^c = (\mathbf{W}^c \mathbf{s}_t^u + \mathbf{b}^c)^\top \mathbf{E}_t^u, \quad p(c_t|\mathbf{s}_t^u, a_t) = \exp(\mathbf{V}_i^c) / \sum_{j=1}^{|a_t|} \exp(\mathbf{V}_j^c) \qquad \text{------(2)}$$

The reward is calculated by,

$$r_t(\mathbf{s}_t^u, a_t) = f_r\big((\mathbf{W}^r \mathbf{s}_t^u + \mathbf{b}^r)^\top \mathbf{e}_t^u\big), \qquad \text{------(3)}$$

The user behavior model is estimated by maximum likelihood estimation.

Agent Model

Similar to that on the user side, given the projected click vectors, we model states on the agent side as,

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

Based on the current state, 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:

$$\pi(i \in a_t | \mathbf{s}_t^a) = \frac{\exp(\mathbf{W}_i^a \mathbf{s}_t^a + \mathbf{b}_i^a)}{\sum_{j=1}^{|C|} \exp(\mathbf{W}_j^a \mathbf{s}_t^a + \mathbf{b}_j^a)}, \quad \text{------(2)}$$

Adversarial Policy Learning

- Training of User Model from offline data introduces inherent bias from the observations specific modeling choices. The bias affects the sequence generation and thus may cause biased value estimation.
- To reduce the effect of bias, we apply **adversarial training** to control the training of both User Model and Agent.
- The discriminator is also used to rescale the generated rewards for policy learning. Therefore, the learning of agent A considers both sequence generation and target rewards.

Adversarial Training

A discriminator D is used to evaluate when given sequence, the probability that is generated from the real recommendation environment.

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,

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

Policy Learning - REINFORCE

REINFORCE (Monte-Carlo policy gradient) relies on an estimated return by Monte-Carlo methods using episode samples to update the policy parameters.

Gradient estimation for User

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

Gradient estimation for Agent

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

Implementation

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**

Implementation details

- **Hardware**: GCP with GPU Nvidia V100, Google Colab Pro with GPU Nvidia P100
- **Platform**: Mac OS; Package Management: Anaconda
- **Framework:** Python 3.5+ , Numpy 1.18.0 , Tensorflow 1.15.0, Pytorch 1.1.0
- **Functionalities:** Generates recommendation which maximizes short term rewards without compromising the long term rewards by using RL and adversarial network.
- **Limitations:** Agent only as good as the model learnt, Also, sometimes this becomes a bottleneck, as the model becomes surprisingly tricky to learn. Computationally more complex than model-free methods.

Experimental Evaluation

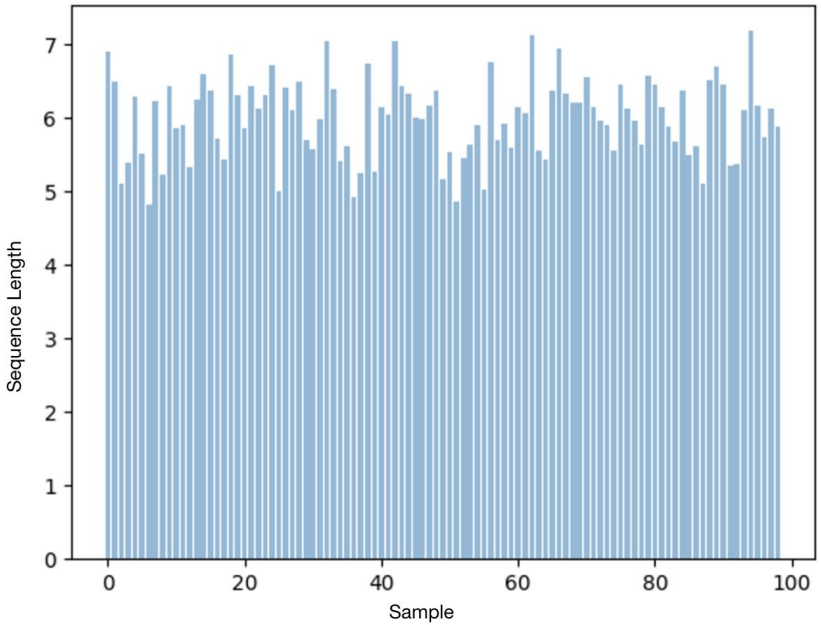
Online: Simulator

- Recommends the top-20 items with highest rewards
- Simulator generates clicks proportional to rewards
- 8000/1000/1000 sessions for training/ validation/ testings.

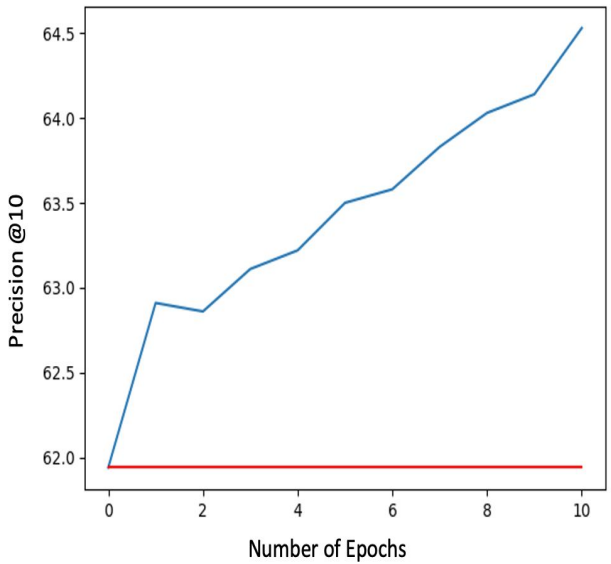
Offline: CIKM 2016 Dataset

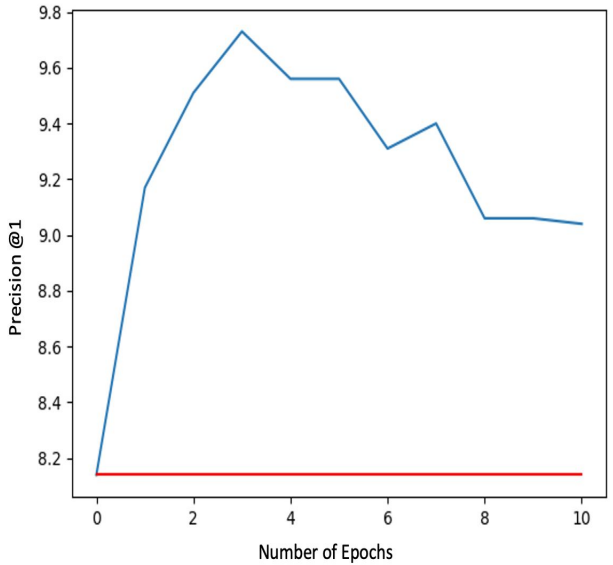
- Personalized E-Commerce Search Challenge.
- 65,284/1,718/1,720 sessions for training/validation/testing.

Sample distribution of simulated dataset



Simulation Results





Results

Model	Baseline	IRecGAN	
	LSTM	Simulation (top-k)	CIKM
Precision@10(%)	32.89	**64.06**	**34.06**
Precision@1(%)	8.20	8.06	7.23

Results

- With the help of adversarial training, IRecGAN achieved encouraging P@10 improvement against the baseline. This verifies the effectiveness of reinforcement learning, especially its adversarial training strategy for utilizing the offline data with reduced bias.
- The adversarial training solution helped to improve the user behavior model

Conclusion

- Developed a practical solution for utilizing offline data to build a model-based reinforcement learning solution for recommendation.
- We introduce adversarial training for joint user behavior model learning and policy update.
- Our evaluations in both simulated and real-world recommendation datasets verify the effectiveness of our solution.

Github Link

https://github.com/aathiramanoj/rl_rec_gan

Thank You

Q & A