Off-policy Recommendation System without Exploration

Chengwei Wang, Tengfei Zhou, Chen Chen, Tianlei Hu, Gang Chen
Recommendation as Reinforcement Learning Problem

Markov Decision Process (MDP)

- **State Space**: The state contains chronological clicked items of user $u$, i.e., $s_t^u = \{i_1, \ldots, i_n\}$

- **Action space** $\mathcal{A}$: The action space is the item set

- **Transition probability** $P(s_{t+1}^u | s_t^u, a_t^u)$:

$$s_{t+1}^u = \begin{cases} s_t^u \cup \{a_t^u\} & \text{if user } u \text{ clicks item } a_t^u \\ s_2^u & \text{otherwise} \end{cases} \quad (1)$$

- **Reward** $r(s_t^u, a_t^u)$:

$$r(s_t^u, a_t^u) = \begin{cases} 1 & \text{if user } u \text{ clicks item } a_t^u \\ 0 & \text{otherwise} \end{cases} \quad (2)$$
Q-Learning

Optimal Strategy

- The Recommender aims to construct a optimal policy \( \pi: S \rightarrow \mathcal{A} \) which optimize cumulative rewards

- The optimal policy is the maximizer of optimization problem:

\[
\max_\pi \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t))
\]  

where \( \gamma \in [0, 1] \) controls the balance between immediate and long term reward

Bellman Equation

- According to MDP, the solution of problem (1) is:

\[
\pi(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)
\]

where \( Q(\cdot, \cdot) \) is the Q-function satisfying Bellman Equation:

\[
Q(s, a) = r(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{a'} Q(s', a')
\]  

Q-Learning

- Bellman Equation (3) can be solved by the following occurrence

\[
Q^{k+1}(s, a) = r(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a} Q^k(s', a)
\]

- Let \( B = \{(s^u_t, a^u_t, s^u_{t+1}, r^u_t)\} \) be the interaction dataset between recommender and environment

- By Monte Carlo method, iteration (4) can be approximated via:

\[
Q^{k+1}(s_t, a_t) = r_t + \gamma \max_{a} Q^k(s_{t+1}, a)
\]

where \((s_t, a_t, s'_{t+1}, r_t) \in B\). The above update is often dumbed as Q-learning

Shortcomings of Q-Learning

- Q-learning may have unrealistic estimate on unobserved state-action pairs, which lead to over optimistic or pessimistic decision and makes the performance of an recommender unstable

- To fix such instability, the recommender has to interaction with online users continually.
Kumar et al. (2019) propose the Batch Constrained Q-Learning (BCQ) method. BCQ avoids exploration error by explicitly constraining an agent’s candidate actions in the training set which result in learning process:

$$Q^{k+1}(s_t, a_t) = r_t + \gamma \max_{(s_{t+1}, a) \in B} Q^k(s_{t+1}, a)$$

(1)

Due to the sparsity of recommendation dataset, BCQ update (4) can usually be simplified to

$$Q^{k+1}(s_t, a_t) = r_t + \gamma Q^k(s_{t+1}, a_{t+1})$$

(2)

Such iteration implicitly assumes that the observed action $$a_{t+1}$$ optimal for state $$s_{t+1}$$, which is impractical.

GCQ utilizes a neural generator $$g_\theta(a|s)$$ to recover the distribution of observed dataset.

Then, the Q-function is updated on a candidate set sampled from the generator.

the main iteration of GCQ

$$\begin{align*}
A^k &= \{a_i \sim g_\theta(a|s_{t+1})\}_{i=1}^c \\
Q^{k+1}(s_t, a_t) &= r_t + \gamma \max \{Q^k(s_{t+1}, a) | a \in A^k\}
\end{align*}$$

(3)

Deep-GCQ: One can use a deep neural network $$Q_\theta(s, a)$$ to approximate the unknown Q-function.
Architecture of State Encoder

The Encoder

- The embedding layer maps a user or an item into correspondent semantic vector
- The Recurrent layer transforms the click sequence into hidden states
- The Attention layer aggregate hidden states into a feature vector

The Q-Net

- Considering that the optimal action shall have close correlations with the current state, we use the inner product of the two object’s feature vectors to model the Q-function:
  \[ Q_\theta(s, a) = (e)^T q_a \]
- where \( q_a \) is item \( a \)'s embedding vector
The Huffman tree is built according to the popularity of items.

- We assign Huffman code to each node of the tree by the following rules:
  - Encode the root by $b_0 = 0$
  - For a node with code $b_0 b_1 ... b_j$, encode its left child by $b_0 b_1 ... b_{j-1} 0$
  - And right child by $b_0 b_1 ... b_{j-1} 1$

- For node $b_0 b_1 ... b_j$ (abbreviated by $b_{0:j}$), let $z_{b_{0:j}} \in \mathbb{R}^d$ be its embedding vector.

- Suppose item $a$ is encoded by $b_{0:j}$, its generating probability is modeled by:

$$ g_\theta(a|s) = \prod_{k=0}^{j-1} \frac{(\sigma(z_{b_{0:k}}^T e))^{b_{k+1}} (1 - \sigma(z_{b_{0:k}}^T e))^{1-b_{k+1}}}{\sigma^{(k)}(z_{b_{0:k}}^T e)} $$

(1)

where $\sigma(\cdot)$ is the sigmoid function, $e$ is the feature vector of state $s$.

- Sampling an item $a$ involves calculating $j$ sigmoid units which takes $O(jd)$ flops, where $j = O(\log |A|)$ is the height of the Huffman Tree.
Training Algorithm

- Loss function of the Generator

\[ nll(\theta) = -\frac{1}{|B|} \sum_{(s,a) \in B} \log g_\theta(a|s). \]

- Loss function of the Q-net

\[ qloss(\theta) = (Q_\theta(s,a) - r + \gamma \max \{ Q_\theta(s', a) | a \in A \})^2 \]

- Joint Loss

\[ \min_{\theta} qloss(\theta) + \lambda nll(\theta) \quad (1) \]

Algorithm 1: Generator Constrained Deep Q-Learning

```
input: Replay Buffer B, size of candidate set c,
regularizer \( \lambda \), number of iterations \( K \),
discount rate \( \gamma \), learning rate \( \eta \)

// Build tree and Initialize Networks
1 tree = BuildHoffmanTree( B )
2 \( \theta_0 \) = InitializeParameters( tree )

3 for ( \( k = 0; k < K; k++ \) ) do
   // sample a tuple from dataset
   4 (s, a, s', r) = GetRandomSample( B )
   // estimate current Q value
   5 \( A = \{ a_i | a_i \sim g_\theta(a|s), i \leq c \} \)
   6 \( \hat{Q} = r + \gamma \max \{ Q_\theta(a_i, s') | a_i \in A \} \)
   // compute stochastic joint loss
   7 qloss = \( \frac{1}{2} (\hat{Q} - Q_\theta(s, a))^2 \)
   8 nll = \( -\log g_\theta(a|s) \)
   9 jointloss = qloss + \lambda nll
   // update parameters
   10 d\( \theta_k \) = \( (Q_\theta(s, a) - \hat{Q}) \nabla Q_\theta(s, a) - \frac{\lambda}{g_\theta(a|s)} \nabla g_\theta(a|s) \)
   11 \( \theta_{k+1} = \theta_k - \eta d\theta_k \)
4 end
```
## Experiments: Offline Evaluation

### Offline recall@k

|       | M1M       | M10M      | AMZ       |
|-------|-----------|-----------|-----------|
|       | Reca@1    | Reca@5    | Reca@10   | Reca@1    | Reca@10   | Reca@1    | Reca@5    | Reca@10   |
| DQN   | 0.0088    | 0.0314    | 0.0770    | 0.0052    | 0.0248    | 0.0429    | 0.0445    | 0.1877    | 0.3091    |
| GRU4Rec | 0.0079    | 0.0308    | 0.0540    | 0.0054    | 0.0235    | 0.0373    | 0.2735    | 0.4568    | 0.543     |
| MF    | 0.0086    | 0.0324    | 0.0561    | 0.0074    | 0.0262    | 0.0439    | 0.2517    | 0.4359    | 0.5224    |
| W&D   | 0.0069    | 0.0313    | 0.0519    | 0.0055    | 0.0238    | 0.0389    | 0.3734    | 0.5405    | 0.5982    |
| DEERS | 0.0048    | 0.0257    | 0.0461    | 0.0037    | 0.0193    | 0.0373    | 0.2926    | 0.6013    | 0.7176    |
| DDPG  | 0.0083    | 0.0353    | 0.0596    | 0.0045    | 0.0210    | 0.0344    | 0.2359    | 0.4160    | 0.4743    |
| GCQ   | **0.0110** | **0.0495** | **0.0897** | 0.0054    | **0.0270** | **0.0539** | **0.3764** | **0.6015** | **0.6747** |

### Offline precision@k

|       | M1M       | M10M      | AMZ       |
|-------|-----------|-----------|-----------|
|       | Prec@1    | Prec@5    | Prec@10   | Prec@1    | Prec@5    | Prec@10   |
| DQN   | 0.1543    | 0.1462    | 0.1396    | 0.0734    | 0.0754    | 0.0722    | 0.0523    | 0.0489    | 0.0432    |
| GRU4Rec | 0.1223    | 0.1043    | 0.0910    | 0.0922    | 0.0732    | 0.0588    | 0.3309    | 0.1263    | 0.0798    |
| MF    | 0.1187    | 0.0920    | 0.0807    | **0.1207** | 0.0907    | 0.0695    | 0.3133    | 0.1220    | 0.0779    |
| W&D   | 0.0992    | 0.0862    | 0.0740    | 0.1074    | 0.0836    | 0.0641    | 0.4539    | 0.1559    | 0.0920    |
| DEERS | 0.0770    | 0.0789    | 0.0757    | 0.0539    | 0.0582    | 0.0585    | 0.3414    | 0.1680    | 0.1110    |
| DDPG  | 0.1598    | 0.1313    | 0.1155    | 0.0727    | 0.0679    | 0.0580    | 0.3016    | 0.1277    | 0.0791    |
| GCQ   | **0.1789** | **0.1658** | **0.1547** | 0.0930    | **0.0931** | **0.0930** | **0.4703** | **0.1829** | **0.1092** |

### Computational time

|       | M1M | M10M | AWZ |
|-------|-----|------|-----|
| DQN   | 116.2 (s) | 1175.7 | 139.9 |
| DDPG  | 120.5 | 1219.2 | 145.2 |
| DEER  | 129.1 | 1306.3 | 155.5 |
| GCQ   | **86.10** | **870.9** | **103.7** |
Experiments: Online Simulation

Cumulative rewards in simulated online environment
Conclusion

• We proposed a novel Generator Constrained Q-learning technique for recommendation tasks

• GCQ stably learns recommendation policy from offline dataset without further interaction with online environment

• We devise a novel generator based on Huffman Tree to reduce decision time complexity

• Empirical results show that GCQ outperforms state-of-the-art methods
Thank you for your listening