Social Attentive Deep Q-network for Recommendation

Yu Lei, Zhitao Wang, Wenjie Li
The Hong Kong Polytechnic University
Hong Kong, China
{csylei,csztwang,cswjli}@comp.polyu.edu.hk

Hongbin Pei
Jilin University
Changchun, China
peihb15@mails.jlu.edu.cn

ABSTRACT

While deep reinforcement learning has been successfully applied to recommender systems, it is challenging and unexplored to improve the performance of deep reinforcement learning recommenders by effectively utilizing the pervasive social networks. In this work, we develop a Social Attentive Deep Q-network (SADQN) agent, which is able to provide high-quality recommendations during user-agent interactions by leveraging social influence among users. Specifically, SADQN is able to estimate action-values not only based on the users' personal preferences, but also based on their social neighbors' preferences by employing a particular social attention layer. The experimental results on three real-world datasets demonstrate that SADQN significantly improves the performance of deep reinforcement learning agents that overlook social influence.

CCS CONCEPTS

• **Information systems → Recommender systems**; • **Computing methodologies → Reinforcement learning**.

KEYWORDS

DQN; reinforcement learning; recommendation; social networks

ACM Reference Format:
Yu Lei, Zhitao Wang, Wenjie Li and Hongbin Pei. 2019. Social Attentive Deep Q-network for Recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19), July 21–25, 2019, Paris, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3331184.3331302

1 INTRODUCTION

Reinforcement learning aims at learning an agent that can autocontrol its behavior in an environment, in order to achieve a goal [12]. By integrating both reinforcement learning and deep neural networks, deep reinforcement learning agents have shown human-level or even better performance in some complex problems such as playing Atari [9] and Go [11]. Recently, some researchers incorporated the ideas and techniques of deep reinforcement learning into recommender systems, and proposed several novel recommendation algorithms which have shown great potential in a variety of recommendation domains [3, 15, 17].

SIGIR '19, July 21–25, 2019, Paris, France

ACM ISBN 978-1-4503-6172-9/19/07…$15.00
https://doi.org/10.1145/3331184.3331302

Successful as they are, the existing reinforcement learning based recommendation approaches only exploit the user-item feedback data. However, with the emergence of online social networks such as Twitter, additional social information of users is usually available to the recommender. According to the social influence theory, users are influenced by others in the social network, leading to the homophily effect that social neighbors may have similar preferences [1]. Thus, it is a potential way to improve recommendation quality by leveraging social influence among users. While this problem has been widely studied in traditional social recommendation domains [7, 13], it has not been investigated in the context of deep reinforcement learning.

In this work, we develop a novel deep reinforcement learning agent, termed Social Attentive Deep Q-network (SADQN), which is able to leverage social influence to improve the quality of recommendations during the interactions with users. The key idea is that we estimate the action-values by using a combination of two action-value (Q) functions, the *personal* action-value function Q^P and the *social* action-value function Q^S (see Figure 1). Intuitively, Q^P estimates action-values based on users' personal preferences, as most of the existing methods do. In contrast, Q^S is able to estimate action-values based on their social neighbors' preferences, by utilizing a particular *social attention* layer. By integrating both functions, SADQN is able to autonomously learn effective recommendation policies that take advantage of both personal preferences and social influence. As a result, it is capable to produce high-quality interactive recommendations that achieve maximal long-term rewards in social recommendation scenarios. We verify its capability by conducting solid experiments on three real-world datasets. The results show that it remarkably outperforms two state-of-the-art deep reinforcement learning agents that fail to consider social influence, as well as several traditional recommendation methods.

2 PRELIMINARY

We consider a recommender system with user set $\mathcal{U} = \{1, ..., m\}$ and item set $\mathcal{I} = \{1, ..., n\}$. Let $R \in \mathbb{R}^{m \times n}$ be the user-item feedback matrix, where $R_{ui} = 1$ if user u gives a positive feedback on item i (clicks, watches, etc.), and $R_{ui} = 0$ otherwise. Let $S \in \mathbb{R}^{m \times m}$ be the adjacent matrix of a social network among the same users, where $S_{uv} = 1$ if user u has a positive relation to user v (follows, trusts, etc.), and $S_{uv} = 0$ otherwise. In this paper, we study an interactive recommendation problem, which can be formulated as a standard reinforcement learning task [12]. Specifically, an agent (recommender) and an environment (target user u) interact at discrete time steps. At each time step t, the agent observes the environment's state s_t (representing the current preferences of user u), and accordingly takes an action (item) a_t based on its policy (probability distributions over actions given states). One time step later, as a consequence of its action, the agent receives a reward

r_{t+1} (R_{ua_t}) and next state s_{t+1} from the environment. The goal of the agent is to maximize the cumulative reward in T interactions.

To learn the optimal policy, an effective way is to approximate the optimal action-value function Q^* by using a neural network function approximator (known as Q-network), i.e., $Q(s, a; \theta) \approx Q^*(s, a)$ [9]. The Q-network can be trained by performing Q-learning updates based on agent's experiences (s, a, r, s'). Specifically, the gradient of the loss function w.r.t. the network weights is given by:

$$\nabla_\theta L(\theta) = \mathbb{E}_{s,a,r,s'} \left[(y - Q(s, a; \theta)) \nabla_\theta Q(s, a; \theta) \right], \quad (1)$$

where $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$ is the target for current iteration, γ is the discount factor, and θ^- are the network parameters from the previous iteration, which are held fixed when optimizing the loss function $L(\theta)$. In practice, rather than computing the full expectations in the above gradient, a more convenient way is to perform stochastic gradient descent on sampled transitions (s, a, r, s').

3 SOCIAL ATTENTIVE DEEP Q-NETWORK

The basic idea behind SADQN is that the action-value $Q(s_t, a)$ is estimated by the combination of two action-value functions: $Q(s_t, a) = Q^P(s_t, a) + Q^S(s_t, a)$, where $Q^P(s_t, a)$ and $Q^S(s_t, a)$ denote the *personal* action-value function and the *social* action-value function, respectively. Intuitively, Q^P estimates action-values based on user u's personal preferences, while Q^S estimates action-values based on his/her social neighbors' preferences. Next, we will orderly describe the architecture and training algorithm of SADQN.

3.1 The Architecture of SADQN

We assume that the state s_t is a f-dimensional feature vector $U_u^t \in \mathbb{R}^f$, denoting the real-time preferences of target user u at time step t. For each user $v \in \mathcal{U}$, there is a f-dimensional feature vector $U_v \in \mathbb{R}^f$, denoting the overall preferences of user v observed in advance. For each item (action) $a \in \mathcal{I}$, there is also a f-dimensional feature vector $V_a \in \mathbb{R}^f$, denoting the overall features of item a. Let $\mathcal{N}(u) = \{v : S_{uv} = 1\}$ denote the set of social neighbors that user u trusts/follows in the social network. In our experiments, the feature matrices $U \in \mathbb{R}^{f \times m}$ and $V \in \mathbb{R}^{f \times n}$ are pre-trained by a standard matrix factorization model [5] together with a negative sampling technique [10] based on feedback data R, which are held fixed during the user-agent interactive process. The target user's vector U_u^t is updated by performing online matrix factorization on the real-time feedback data R_{ua_t}:

$$U_u^{t+1} \leftarrow U_u^t + \beta \left(R_{ua_t} - (U_u^t)^{\mathrm{T}} V_{a_t} \right) V_{a_t}, \quad (2)$$

where β is the learning rate. Based on these notations and definitions, we illustrate the architecture of SADQN in Figure 1.

The right part of SADQN is the personal action-value function approximator $Q^P(s_t, a; \theta^P)$, which is a standard 4-layer multilayer perceptron (MLP). It takes the concatenation of user vector U_u^t (i.e., the features of state s_t) and item vector V_a (i.e., the features of action a) as input, followed by two fully connected (FC) layers with ReLU activation, and outputs the personal action-value $Q^P(s_t, a)$.

The left part of SADQN is the social action-value function approximator $Q^S(s_t, a; \theta^S)$, in which there is a *social attention* (SA) layer. The goal of the SA layer is to select influential social neighbors for target user u at time step t, and summarize the neighbors' features to a context vector C_u^t. Then, the concatenation of context vector C_u^t and item vector V_a is used to feed another 4-layer MLP, which will output the social action-value $Q^S(s_t, a)$.

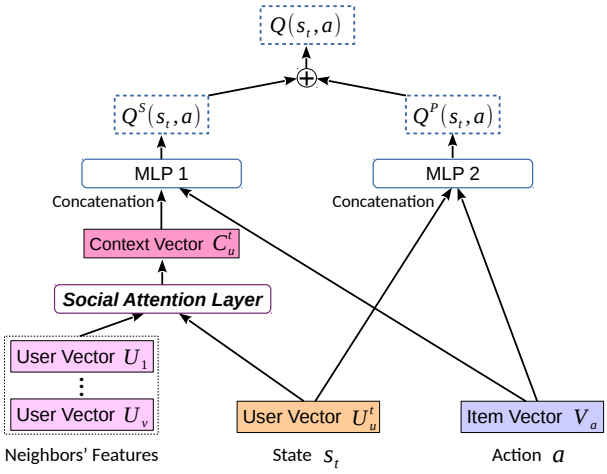


Figure 1: The architecture of SADQN.

Specifically, we compute the context vector C_u^t by the following procedure. We employ DOT product to compute the *attention coefficient* of target user u and his/her social neighbor $v \in \mathcal{N}(u)$:

$$e_{uv}^t = dot(U_u^t, U_v), \quad (3)$$

which indicates the social influence strength of user v to user u at time step t. Similar to [14], we also compute the attention coefficient of target user u and himself/herself by $e_{uu}^t = dot(U_u^t, U_u^t)$. Then, we use softmax function to normalize the attention coefficients: $\alpha_{uv}^t = \frac{\exp(e_{uv}^t)}{\sum_{w \in \mathcal{N}(u)_+} \exp(e_{uw}^t)}$, where $\mathcal{N}(u)_+ = \mathcal{N}(u) \cup \{u\}$. Finally, the context vector C_u^t is computed by: $C_u^t = \alpha_{uu}^t U_u^t + \sum_{v \in \mathcal{N}(u)} \alpha_{uv}^t U_v$.

In our experiments, we also tried several different ways to compute the attention coefficient e_{uv}^t, such as CONCAT:

$$e_{uv}^t = ReLU(dot(\mathbf{w}, concat(U_u^t, U_v))), \quad (4)$$

where $\mathbf{w} \in \mathbb{R}^{2f}$ is the weight vector of a single-layer feedforward network. However, the performance of this approach showed no significant difference with the one in Equation 3. Moreover, we also used a single-layer graph attention network (GAT) [14] to compute the context vector C_u^t. Unfortunately, it did not show comparable performance against the above two approaches (see Table 3).

3.2 Training SADQN

To train SADQN, we employ the popular Q-leaning algorithm, similar to the work of DQN [9]. However, we do not adopt the training tricks *experience replay* and *target network* used by DQN, as they are not able to improve the performance of SADQN for our tasks. To ensure the Q-network converges to optimal Q^* function, sufficient transitions (s, a, r, s') of all possible states and actions are needed for Q-learning updates [12]. To this end, we propose a particular training scheme that enables the agent to collect transitions based on the feedback data of all training users. Specifically, in each episode, we uniformly sample a user u from training set $\mathcal{U}_{train}$ as the current target user, which will interact with the agent and

Algorithm 1: Training SADQN

Input: $\mathcal{U}_{train}$, R, the pre-trained U, V
Output: the trained Q-network Q

Initialize Q with random weights
for *episode* $= 1, N$ **do**
  Uniformly sample a target user u from $\mathcal{U}_{train}$
  Set user vector $U_u^0 = \mathbf{0}$ and initial state $s_0 = \mathbf{0}$
  for $t = 0, T-1$ **do**
    Choose item a_t by the ϵ-greedy policy w.r.t. $Q(s_t, a)$
    Recommend a_t to user u and receive feedback R_{ua_t}
    Get U_u^{t+1} according to **Equation 2**
    Set reward $r_{t+1} = R_{ua_t}$ and state $s_{t+1} = U_u^{t+1}$
    Update Q's weights by performing SGD on the transition $(s_t, a_t, r_{t+1}, s_{t+1})$ (**Equation 1**)
  end
end

Table 1: The statistics of datasets.

Statistics	LastFM	Ciao	Epinions
#users	1,874	7,260	23,137
#items	2,828	11,166	23,585
#observed user feedbacks	71,411	147,799	461,982
#observed social relations	25,174	110,715	372,205

generate corresponding states and rewards. To ensure exploration, in each state s_t, the agent uses a ϵ-greedy policy that selects a greedy action $a_t = \arg\max_a Q(s_t, a)$ with probability $1 - \epsilon$ and a random action with probability ϵ. The full algorithm for training SADQN is presented in Algorithm 1. The training process could last for any number of episodes as long as the Q-network Q is not converged. After training, the agent can be used to make interactive recommendations for any new user v. It only needs to interact with user v as usual, observe states, and always recommend the greedy item with respect to the trained Q at each time step.

4 EXPERIMENTS

4.1 Experimental Setup

We employ three publicly available datasets: LastFM [2], Ciao [13], and Epinions [8] for our experiments. All the datasets contain a user-item feedback matrix and a user-user social network. As we consider the recommendation problem with implicit feedback, we convert the values of all observed feedbacks to 1. Besides, we remove the users or items that have fewer than 5 feedbacks, so as to ensure there is enough data for training and testing. The basic statistics of the obtained datasets are shown in Table 1.

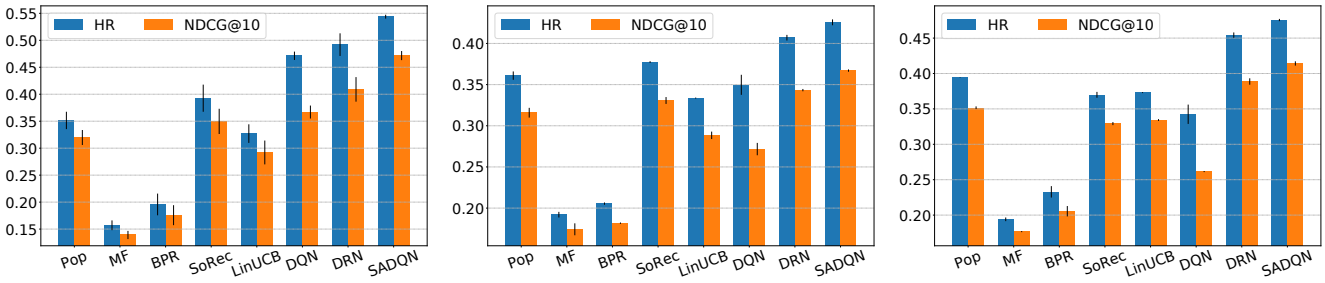
4.1.1 Evaluation Protocol. To simulate the real-world interactive recommendation process, we assume the observed feedbacks in the datasets are unbiased and interactive, similar to the existing works [6, 16]. At each time step t, the agent is asked to recommend an item a_t to target user u, and the reward is determined by the ground-truth value of the feedback in the dataset, i.e., $r_{t+1} = R_{ua_t}$. Similar to [4], we randomly choose 1000 unobserved (u, i) pairs of user u as the negative feedbacks ($R_{ui} = 0$) for each episode. The agent is forced to pick items from the available set that consists of the 1000 negative items and the observed positive items. We adopt two popular evaluation metrics *Hit Ratio* (HR) and *Normalized Discounted Cumulative Gain* (NDCG). The HR indicates the ratio of positive items among the T recommended items in the T-step interactive process. The NDCG_t value is calculated on a ranking list of the available item set at time step t, which is produced according to the agent's predictions (e.g., Q-values). The NDCG is obtained by averaging the NDCG_t values for $t = 0, ..., T-1$. We truncate the ranking list at 10 to compute the NDCG@10 values, and set $T = 20$ for our evaluation. We split each dataset by randomly choosing 10% users who have more than 20 observed feedbacks as the testing set $\mathcal{U}_{test}$, and the remaining users as the training set $\mathcal{U}_{train}$. We conduct each experiment on 5 data splits obtained with different random seeds, and calculate the average results for our evaluation.

4.1.2 Baselines. We compare our agent **SADQN** with a variety of baselines, including two deep reinforcement learning methods **DQN** [9] and **DRN** [17], a contextual bandit method **LinUCB** [6], a social recommendation method **SoRec** [7], a learning to rank method **BPR** [10], a matrix factorization method **MF** [5], and a popularity method **Pop**. To make the baselines applicable to our recommendation problem, we adopt the same state/action features and training scheme of SADQN for DQN, DRN and LinUCB, and use the same negative sampling technique of BPR for MF and SoRec. We also adopt the same hidden layers of the personal action-value function of SADQN, i.e., two FC layers of 256 units with ReLU activations, for DQN and both the value and advantage functions of DRN, which lead to better performance. Moreover, we set the feature dimensionality $f = 64$ for all methods (excluding Pop), and set the number of time steps in training phase, $T_{train} = n_u/2$, for SADQN, DQN, DRN and LinUCB, where n_u denotes the number of observed feedbacks of user u in the dataset. Other hyperparameters of the compared methods are tuned based on cross-validation.

4.2 Experimental Results

The experimental results of all methods, in terms of the mean and standard deviation of both HR and NDCG@10 metrics, are shown in Figure 2. As we can see, the proposed SADQN remarkably outperforms the deep reinforcement learning methods DQN and DRN that fail to consider social influence, in terms of both metrics on all datasets. This demonstrate that social influence is quite beneficial to deep reinforcement learning agents in performing interactive recommendations. The SADQN agent also shows significant advantages over other types of baselines, which validates again its effectiveness and robustness. The traditional model-based methods MF and BPR show poor performance, as no feedback data is available at time step $t = 0$, while the social recommendation method SoRec demonstrates much better performance. Besides, the popularity method Pop is a competitive baseline in the cold-start setting, in spite of its non-personalized characteristics.

4.2.1 The Impact of Social Influence. To analyze the importance of social influence, we also compare the performance of two variants of SADQN. They are SADQN^P and SADQN^S, which estimate action-values by using Q^P and Q^S, respectively (see Figure 1). Note that



(a) LastFM (b) Ciao (c) Epinions

Figure 2: Performance comparison in terms of the mean (bar) and standard deviation (line) of HR and NDCG@10 metrics.

Table 2: The HR results of different variants of SADQN.

Model	LastFM	Ciao	Epinions
SADQN^P	0.4927±0.0211	0.4028±0.0005	0.4522±0.0048
SADQN^S	0.5200±0.0157	0.4105±0.0004	0.4440±0.0033
SADQN	**0.5438±0.0036**	**0.4256±0.0031**	**0.4755±0.0016**

Table 3: The HR results of different attention mechanisms.

Attention	LastFM	Ciao	Epinions
GAT [14]	0.5191±0.0114	0.4179±0.0060	0.4667±0.0018
CONCAT	0.5435±0.0076	0.4247±0.0072	**0.4782±0.0012**
DOT	**0.5438±0.0036**	**0.4256±0.0031**	0.4755±0.0016

SADQN^P is different from the DQN baseline, because DQN only uses state s_t for input and outputs the Q values of all actions in that state. The HR results of SADQN^P, SADQN^S and SADQN are shown in Table 2, where the bold font indicates the best performing model. As we can see from Table 2, the integrated model SADQN performs best on all datasets. The social model SADQN^S also shows better performance than the personal model SADQN^P on LastFM and Ciao datasets, but worse performance on Epinions. This is mainly because the density of social relations in Epinions dataset is extremely low (only 0.09%), which affects the learning performance of SADQN^S on the social influence. The NDCG@10 results show similar trends, which are not reported here due to space limitation.

4.2.2 Comparison of Different Attention Mechanisms. We now compare the performance of different attention mechanisms discussed in Section 3.1. We evaluate three SADQN agents which adopt the attention mechanisms GAT [14], CONCAT (Equation 4) and DOT (Equation 3, i.e., the default one used by SADQN), respectively. The comparison results in terms of HR are shown in Table 3. The two attention mechanisms CONCAT and DOT perform very closely on all datasets, and outperform GAT.

5 CONCLUSIONS

In this work, we developed a Social Attentive Deep Q-network (SADQN) agent, which is able to leverage social influence to improve the quality of interactive recommendations. Specifically, it estimates action-values not only based on the users' personal preferences, but also their social neighbors' preferences by employing a social attention layer. The solid experimental results have shown that SADQN remarkably outperforms two deep reinforcement learning agents that overlook social influence, as well as several traditional recommendation methods.

ACKNOWLEDGMENTS

The work was supported by National Natural Science Foundation of China (61672445) and The Hong Kong Polytechnic University (G-YBP6).

REFERENCES

[1] E. Bakshy, I. Rosenn, C. Marlow, and L. Adamic. 2012. The role of social networks in information diffusion. In *WWW*. ACM, 519–528.
[2] I. Cantador, P. L. Brusilovsky, and T. Kuflik. 2011. *Second workshop on information heterogeneity and fusion in recommender systems (HetRec2011)*. ACM.
[3] S. Chen, Y. Yu, Q. Da, J. Tan, H. Huang, and H. Tang. 2018. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *SIGKDD*. ACM, 1187–1196.
[4] P. Cremonesi, Y. Koren, and R. Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*. ACM, 39–46.
[5] Y. Koren, R. Bell, and C. Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.
[6] L. Li, W. Chu, J. Langford, and R. E. Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *WWW*. ACM, 661–670.
[7] H. Ma, H. Yang, M. R. Lyu, and I. King. 2008. Sorec: social recommendation using probabilistic matrix factorization. In *CIKM*. ACM, 931–940.
[8] P. Massa and P. Avesani. 2007. Trust-aware recommender systems. In *RecSys*. ACM, 17–24.
[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv* (2013).
[10] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*. 452–461.
[11] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.
[12] R. S. Sutton and A. G. Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.
[13] J. Tang, H. Gao, H. Liu, and A. Das Sarma. 2012. eTrust: Understanding trust evolution in an online world. In *SIGKDD*. ACM, 253–261.
[14] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. 2017. Graph attention networks. *arXiv* (2017).
[15] X. Zhao, L. Zhang, Z. Ding, L. Xia, J. Tang, and D. Yin. 2018. Recommendations with Negative Feedback via Pairwise Deep Reinforcement Learning. In *SIGKDD*. ACM, 1040–1048.
[16] X. Zhao, W. Zhang, and J. Wang. 2013. Interactive collaborative filtering. In *CIKM*. ACM, 1411–1420.
[17] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li. 2018. DRN: A deep reinforcement learning framework for news recommendation. In *WWW*. IW3C2, 167–176.