Social Attentive Deep Q-networks for Recommender Systems

Yu Lei, Zhitao Wang, Wenjie Li, Hongbin Pei, and Quanyu Dai

Abstract—Recommender systems aim to accurately and actively provide users with potentially interesting items (products, information or services). Deep reinforcement learning has been successfully applied to recommender systems, but still heavily suffer from data sparsity and cold-start in real-world tasks. In this work, we propose an effective way to address such issues by leveraging the pervasive social networks among users in the estimation of action-values (Q). Specifically, we develop a Social Attentive Deep Q-network (SADQN) to approximate the optimal action-value function based on the preferences of both individual users and social neighbors, by successfully utilizing a social attention layer to model the influence between them. Further, we propose an enhanced variant of SADQN, termed SADQN++, to model the complicated and diverse trade-offs between personal preferences and social influence for all involved users, making the agent more powerful and flexible in learning the optimal policies. The experimental results on real-world datasets demonstrate that the proposed SADQNs remarkably outperform the state-of-the-art deep reinforcement learning agents, with reasonable computation cost.

Index Terms—DQN, reinforcement learning, recommender systems, social networks.

1 Introduction

Recommender systems aim to accurately and actively provide users with potentially interesting items (products, information or services), which have become a powerful tool to improving user experience and business profit in many online applications such as E-commerce, social networking and video sites. Online recommender systems are intrinsically interactive, in the sense that the recommender and target user usually interacts in a multi-step recommendation process. At each step, the recommender makes a decision to recommend some items to the user, and receives feedback from him/her. Normally, the decision made at current step affects not only the immediate reward, but also the future rewards at later steps. For example, an item recommended currently may not be liked by the user, but the received feedback provides valuable information about his/her interests, which can help the recommender make better recommendations in future [1].

Unfortunately, the majority of research on recommender systems is based on supervised learning, which focuses on learning accurate predictive models from historical feedback data for only single-step recommendations [2], [3], [4], [5]. Most of these recommendation approaches cannot provide a satisfactory solution to the multi-step interactive recommendation problem in real-world scenarios.

To address this issue, a potential way is to use reinforcement learning, which aims at learning an agent that can auto-control its behavior in an environment, in order to achieve a goal [6]. By integrating both reinforcement learning and deep neural networks, deep reinforcement learning agents have shown human-level or even better performance in solving many complex decision making problems such as playing Atari [7], [8] and Go [9]. Recently, a number of 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 [10], [11], [12], [13]. Compared to traditional recommenders, a notable merit of reinforcement learning agents is that they are able to actively discover users' interests through user-agent interactions and recommend items that may bring maximal future rewards.

Successful as they are, the existing reinforcement learning based approaches only exploit user-item feedback data, whose recommendation performance may be greatly reduced when data sparsity and cold-start[1] occur. Fortunately, 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 [14], [15]. Thus, it is a potential way to improve the quality of recommendations by leveraging available social networks, which has been widely studied and demonstrated in traditional social recommendation domains [16], [17], [18], [19], [20], [21]. However, most of the existing social recommendation models cannot be directly extended to reinforcement learning based systems, as they are based on the paradigm of supervised learning.

To the best of our knowledge, this paper makes the first attempt to improve the performance of deep reinforcement learning based recommenders, by effectively utilizing available social networks. In the context of deep reinforcement

- *Y. Lei, Z. Wang, W. Li and Q. Dai are with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China.*
  E-mail: {csylei, csztwang, cswjli, csqydai}@comp.polyu.edu.hk.
- *H. Pei is with the School of Computer Science and Technology, Jilin University, Changchun, China.*
  E-mail: peihb15@mails.jlu.edu.cn.

1. In this work, when discussing cold-start, we always refer to the user cold-start problem.

learning, the performance of many agents is determined by the estimation of the *long-term rewards*, e.g., action-values, which indicate, in the long run, how many benefits the agent will obtain if it recommends the items at current time [6]. Similar to the prediction of *short-term rewards* (e.g., ratings) in traditional recommendation domains, it is biased and inefficient to estimate the action-values based on only users' own preferences, due to the issues of cold-start and data sparsity. Thus, we propose to leverage social neighbors' preferences to promote the estimation of action-values.

To implement this idea, we develop a novel deep reinforcement learning agent, termed Social Attentive Deep Q-network (SADQN). The key idea is that we estimate the action-values by a linear combination of two action-value functions, the *personal action-value function* Q^P and the *social action-value function* Q^S (see Figure 2). Intuitively, Q^P estimates action-values based on target user's personal preferences, as most of the existing methods do. In contrast, Q^S is able to estimate action-values based on his/her social neighbors' preferences, by utilizing an *attention mechanism* to model the influence from different social neighbors to target user. By integrating both functions, SADQN is able to learn recommendation policies that take advantage of both personal preferences and social influence.

While SADQN is straightforward and easily understandable, the simple linear combination of function approximators Q^P and Q^S is not able to model the complicated and diverse trade-offs between personal interests and social influence for all involved users, which limits the performance in approximating the optimal action-value function Q^*. To handle this challenge, we propose an enhanced variant of SADQN, termed SADQN++ (see Figure 3), to fuse Q^P and Q^S more deeply by learning appropriate trade-offs from data autonomously. More specifically, we leverage Q^P and Q^S to learn some relevant hidden representations from personal and neighbors' preferences. Then, we employ additional neural layers to summarize valuable features from these hidden representations, and predict the final action-value based on the summarized features. This way provides more flexibility to SADQN++ in modeling the trade-offs, leading to stronger capability in approximating the optimal action-value function. As a result, SADQN++ is able to learn the optimal policies more effectively.

We empirically validate the performance of the proposed SADQNs by conducting solid experiments on three real-world datasets. The results show that they remarkably outperform four state-of-the-art deep reinforcement learning agents that fail to consider social influence, as well as several traditional recommendation methods. In particular, the relative improvements of SADQN++ against the best performing baseline are at least larger than 8.5% for cold-start recommendation, and 3.5% for warm-start recommendation. More importantly, the significant improvements of SADQNs over the state-of-the-art agents are accomplished with reasonable computation cost.

The remainder of this paper is organized as follows. In Section 2, we formalize the problem of T-step interactive recommendation, and introduce the background of reinforcement learning and Deep Q-network. Then, we present the major content of SADQNs in Section 3. We describe our experimental design, and show the results and analysis in Section 4. We review the related work in Section 5. Finally, we conclude this work in Section 6.

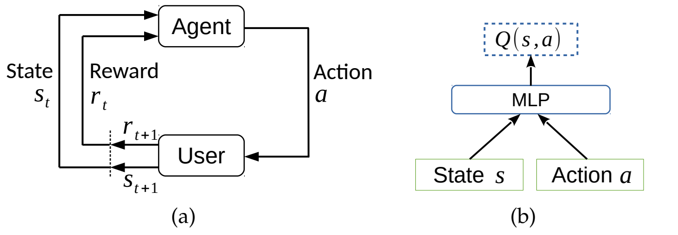


Fig. 1. (a) The user-agent interaction in reinforcement learning based recommender systems. (b) The basic architecture of DQN.

2 Preliminaries

2.1 Problem Formulation

We consider a recommender system with user set $\mathcal{U} = \{1, ..., m\}$ and item set $\mathcal{I} = \{1, ..., n\}$. Let $R \in \mathbb{R}^{m \times n}$ denote the user-item feedback matrix, where $R_{ia} = 1$ if user i gives a positive feedback on item a (clicks, watches, etc.), and $R_{ia} = 0$ otherwise.

We focus on the task of recommending items in sequential user-agent interactions, which can be formulated as a standard reinforcement learning (RL) problem [6]. Specifically, a recommendation agent and a user (an environment that describes the interactive recommendation process of the user) interact at discrete time steps (see Figure 1a). At each time step t, the agent observes the environment's state s_t (representing the current preferences of user i), 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_{ia_t}) and next state s_{t+1} from the environment. The goal of the agent is to maximize the cumulative reward it receives in T interactions.

More formally, the environment (i.e., the user's interactive recommendation process) can be mathematically described by a Markov decision process (MDP), which satisfies the Markov property:

$$\begin{aligned} &Pr[s_{t+1} = s', r_{t+1} = r | s_0, a_0, r_1, ..., s_{t-1}, a_{t-1}, r_t, s_t, a_t] \\ &= Pr[s_{t+1} = s', r_{t+1} = r | s_t, a_t], \end{aligned}$$

for all s', r, and histories $s_0, a_0, r_1, ..., s_{t-1}, a_{t-1}, r_t, s_t, a_t$. This provides well guarantees for applying standard RL methods to solve the MDP [6], that is, to estimate an optimal policy π^* that maximizes the cumulative reward received during the interactive recommendation process.

2.2 Reinforcement Learning

Normally, the agent's policy is a mapping from states to actions, $a = \pi(s)$, or probability distributions over actions given states, $\pi(a|s)$. To learn the optimal policy, the basic idea of many reinforcement learning methods is to estimate the action-value (Q) function, which is formally defined as:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum\nolimits_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right], \quad (1)$$

where γ is the discount factor that balances the importance between future rewards and immediate rewards. It indicates, in the long run, how good it is to take action a in state s while following policy π in future steps. The optimal

Q function, $Q^*(s,a) = \max_\pi Q^\pi(s,a)$, obeys an important identity known as the Bellman optimality equation [22]:

$$Q^*(s,a) = \mathbb{E}_{s' \sim Pr(s'|s,a)} \left[r + \gamma \max_{a'} Q^*(s',a') \right]. \quad (2)$$

Based on the above equation, a popular reinforcement learning algorithm, Q-learning [23], estimates a Q function (i.e., a lookup table) via the following online update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[y - Q(s,a) \right], \quad (3)$$

where $y = r + \gamma \max_{a'} Q(s',a')$ is usually called Q-learning target, α is the learning rate, and (s,a,r,s') denotes a transition the agent experiences during its interactions with the environment. The estimated Q function will converge to Q^* when the number of iterations goes to infinity [6]. The greedy policy w.r.t. Q^* will be an optimal policy π^*.

2.3 Deep Q-network

The classical Q-learning algorithm must maintain a lookup table of all state-action pairs, which cannot handle complex reinforcement learning tasks with enormous state and action spaces. Moreover, it estimates the action-value function separately for each sequence, which has no generalization ability to deal with unseen states and actions [6].

To address these issues, a common practice is to approximate the optimal action-value function by using a function approximator, i.e., $Q(s,a;\theta) \approx Q^*(s,a)$, where θ denote the weights to be learned [6]. For instance, Deep Q-network (DQN) [7] employs a deep neural network as the function approximator. The basic form of DQN is illustrated in Figure 1b. It is actually a multilayer perceptron (MLP), which takes state s and action a as input, and outputs the predicted action-value $Q(s,a)$.

The Q-network $Q(s,a;\theta)$ can be trained by performing Q-learning updates based on the agent's experiences (s,a,r,s'). Specifically, the loss function that needs to be minimized is defined as:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'} \left[\left(y - Q(s,a;\theta) \right)^2 \right], \quad (4)$$

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

$$\theta \leftarrow \theta + \alpha \left[y - Q(s,a;\theta) \right] \nabla_\theta Q(s,a;\theta). \quad (5)$$

3 Social Attentive Deep Q-networks

As mentioned before, it is biased and inefficient to estimate the optimal Q^* function (corresponding to an optimal recommendation policy) only based on individual users' own feedbacks, due to the issues of cold-start and data sparsity. In this section, we propose a novel class of deep reinforcement learning (RL) agents, which are able to estimate Q^* more effectively, by leveraging the available social network among users. Let $S \in \mathbb{R}^{m \times m}$ be the adjacency matrix of the social network, where $S_{ij} = 1$ if user i has a positive relation to user j (follows, trusts, etc.), and $S_{ij} = 0$ otherwise. Let $\mathcal{N}(i) = \{j : S_{ij} = 1\}$ denote the set of social neighbors whom user i trusts/follows.

In what follows, we first describe a state/action representation method that utilizes matrix factorization (MF) to learn high-level vector representations of states and actions. Then, we present a basic Social Attentive Deep Q-network (SADQN), and an enhanced variant, named SADQN++. After that, we propose an enhanced representation method that utilizes a Social MF model, instead of the standard MF model, to learn the high-level state/action representations. Lastly, we describe the training algorithm of SADQNs.

3.1 MF-based States and Actions

We assume that the state s_t is a f-dimensional feature vector $U_i^t \in \mathbb{R}^f$, denoting the real-time preferences of target user i at time step t. For each user $j \in \mathcal{U}$, there is a f-dimensional feature vector $U_j \in \mathbb{R}^f$, denoting the overall preferences of user j 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. The feature matrices $U \in \mathbb{R}^{f \times m}$ and $V \in \mathbb{R}^{f \times n}$ are trained by standard matrix factorization (MF) [24] together with negative sampling [25] based on the historical feedback data R, which are held fixed during the user-agent interactive recommendation process. The target user's vector U_i^t (i.e., state s_t) is initialized as the trained U_i at time step $t = 0$, and is updated by performing online MF on the real-time feedback data R_{ia_t} for each time step t:

$$U_i^{t+1} \leftarrow U_i^t + \alpha \left[\left(R_{ia_t} - (U_i^t)^\top V_{a_t} \right) V_{a_t} - \lambda U_i^t \right], \quad (6)$$

where α is the learning rate, and λ is the L_2 regularization parameter.

In other words, our proposed SADQNs are two-stage pipeline agents, which are different from the typical end-to-end deep RL agents. A typical end-to-end agent directly estimate action-value based on the input of the raw state (the currently observed feedback information of user) and the raw action (the corresponding item id). However, in SADQNs, the action-value prediction is a two-stage procedure. At the first stage, we transform the raw state to a high-level state representation, i.e., the user embedding, by using online MF. At the second stage, the Q-network predicts the action-value based on the MF-based state and the MF-based action (i.e., the item embedding that is trained by offline MF before the user-agent interactive process).

We employ the MF-based state/action representations, rather than the end-to-end trained embeddings, because of two main reasons. (1) The MF-based states capture both the personal preferences of target user and the collaborative relationships (i.e., behavioral similarities) between different users in the shared latent feature space, which will help the Q-network learn more effective recommendation policies. (2) Such a two-stage procedure enables us to take advantage of the well-studied MF models in traditional recommendation domains.

3.2 SADQN: A Linear Fusion Model

The basic idea behind SADQN is that the action-value $Q(s_t,a)$ is estimated by a linear combination of two action-value functions $Q^P(s_t,a)$ and $Q^S(s_t,a)$, which denote the *personal action-value function* and the *social action-value function*, respectively. Intuitively, Q^P estimates the action-values based on target user i's personal real-time preferences, i.e.,

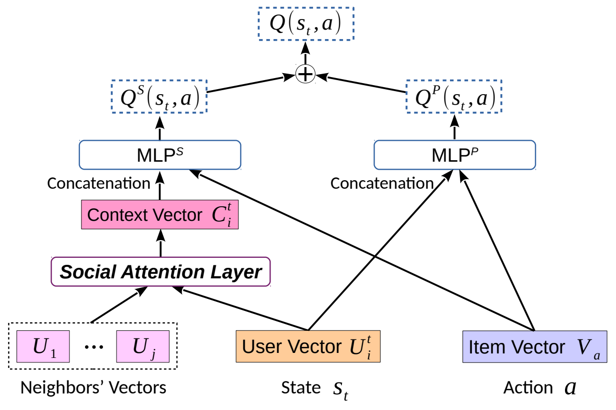


Fig. 2. SADQN: a linear fusion model.

U_i^t. In contrast, Q^S estimates the action-values based on his/her social neighbors' pre-observed preferences, i.e., U_j for $j \in \mathcal{N}(i)$, which are correlated with U_i^t according to the social influence theory.

The architecture of SADQN is illustrated in Figure 2. The right part of SADQN is the personal action-value function approximator Q^P, which is a standard 4-layer MLP. It takes the concatenation of user vector U_i^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, and outputs the personal action-value $Q^P(s_t, a)$. In our experiments, each FC layer consists of 256 neurons with ReLU activation. Therefore, the specific architecture of MLP^P is $2f \rightarrow 256 \rightarrow 256 \rightarrow 1$.

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

Specifically, we compute the context vector C_i^t by the following procedure. We employ the CONCAT attention mechanism to calculate the *attention coefficient* of target user i and his/her social neighbor $j \in \mathcal{N}(i)$:

$$e_{ij}^t = \text{ReLU}(\mathbf{w}^{\mathrm{T}} \cdot \text{CONCAT}(U_i^t, U_j)), \tag{7}$$

where $\mathbf{w} \in \mathbb{R}^{2f}$ is the weight vector of a single-layer feedforward network. The attention coefficient e_{ij}^t indicates the social influence strength of user j to user i at time step t. Similar to [26], we also compute the attention coefficient of user i and himself/herself by: $e_{ii}^t = \text{ReLU}(\mathbf{w}^{\mathrm{T}} \cdot \text{CONCAT}(U_i^t, U_i^t))$. Then, we compute the normalized attention coefficients α_{ij}^t by softmax function:

$$\alpha_{ij}^t = \frac{\exp(e_{ij}^t)}{\sum_{k \in \mathcal{N}(i)_+} \exp(e_{ik}^t)}, \tag{8}$$

where $\mathcal{N}(i)_+ = \mathcal{N}(i) \cup \{i\}$. Finally, we obtain the context vector C_i^t by:

$$C_i^t = \alpha_{ii}^t U_i^t + \sum\nolimits_{j \in \mathcal{N}(i)} \alpha_{ij}^t U_j. \tag{9}$$

In our experiments, we also tried several different attention mechanisms to compute the context vector C_i^t. For example, the attention coefficient e_{ij}^t can be computed by the simple DOT product:

$$e_{ij}^t = \text{DOT}(U_i^t, U_j). \tag{10}$$

However, the performance of this approach showed no significant difference with the one in Equation 7. Moreover, we also implemented a single-layer graph attention network (GAT) [26] to compute C_i^t. Unfortunately, it did not show comparable performance against the above two approaches (see Table 4 for comparison).

To summarize, the action-value $Q(s_t, a)$ estimated by SADQN is formally defined as:

$$\begin{aligned}
\mathbf{h}_1^P &= \text{CONCAT}(U_i^t, V_a), \\
\mathbf{h}_2^P &= \text{ReLU}(\mathbf{W}_2^P \cdot \mathbf{h}_1^P + \mathbf{b}_2^P), \\
\mathbf{h}_3^P &= \text{ReLU}(\mathbf{W}_3^P \cdot \mathbf{h}_2^P + \mathbf{b}_3^P), \\
Q^P(s_t, a) &= (\mathbf{w}_4^P)^{\mathrm{T}} \cdot \mathbf{h}_3^P + \mathrm{b}_4^P; && (11) \\
\mathbf{h}_1^S &= \text{CONCAT}(C_i^t, V_a), \\
\mathbf{h}_2^S &= \text{ReLU}(\mathbf{W}_2^S \cdot \mathbf{h}_1^S + \mathbf{b}_2^S), \\
\mathbf{h}_3^S &= \text{ReLU}(\mathbf{W}_3^S \cdot \mathbf{h}_2^S + \mathbf{b}_3^S), \\
Q^S(s_t, a) &= (\mathbf{w}_4^S)^{\mathrm{T}} \cdot \mathbf{h}_3^S + \mathrm{b}_4^S; && (12) \\
Q(s_t, a) &= (1 - \beta) Q^P(s_t, a) + \beta Q^S(s_t, a). && (13)
\end{aligned}$$

Here, $\mathbf{h}_l^P$, $\mathbf{W}_l^P$ and $\mathbf{b}_l^P$ denote the outputs, trainable weights and biases of l-th layer of MLP^P, respectively. Similar notations are used for MLP^S. $\beta \in [0, 1]$ controls the trade-off between personal preferences and social influence.

The network parameters of SADQN can be trained by performing the Q-learning updates in Equation 5. Note that if we only use Q^P or Q^S to estimate the action-values, SADQN will reduce to a *pure personal model* SADQN^P (which is equivalent to the basic DQN model shown in Figure 1b) or a *pure social model* SADQN^S.

3.3 SADQN++: A Deep Fusion Model

In the previous SADQN model, the personal approximator Q^P and the social approximator Q^S are simply fused by a linear combination at the output level. While this approach is straightforward and easily understandable, such a shallow fusion might limit the performance of Q-network in approximating the optimal action-value function Q^*. For example, the trade-offs between personal interests and social influence may vary considerably, for different users i at different time steps t, or even on different items a.

As such, using the same trade-off parameter β (in Equation 13) for all situations is obviously inappropriate. In fact, we tested different $\beta \in \{0.2, 0.5, 0.8\}$ in our experiments. But the overall performance shows no significant difference, which implies that a good trade-off for one situation may be improper for another. On the other hand, it is infeasible to search or learn an optimal value of β for every situation, e.g., for each state-action pair (s_t, a).

To address this issue, we propose an enhanced variant of SADQN, termed SADQN++, to more deeply fuse the two approximators Q^P and Q^S by autonomously learning good trade-offs from data via additional neural layers.

The architecture of SADQN++ is illustrated in Figure 3, which is quite similar to SADQN, with only differences in the last few layers. Rather than using Q^P and Q^S to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3012346, IEEE Transactions on Knowledge and Data Engineering

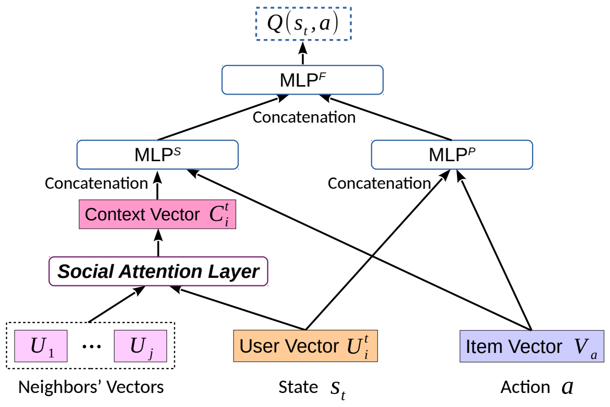


Fig. 3. SADQN++: a deep fusion model.

directly estimate action-values from personal and neighbors' preferences, in SADQN++, we leverage them to learn some hidden vector representations that are relevant to the estimation of action-values. More specifically, MLP^S (MLP^P) in SADQN++ is a 3-layer MLP with the architecture of $2f \rightarrow 256 \rightarrow 256$, which will output the *social hidden representation* (*personal hidden representation*). Then, we employ MLP^F with the architecture of $512 \rightarrow 256 \rightarrow 1$, to autonomously summarize valuable features from both hidden representations, and to predict the final action-value $Q(s_t, a)$ based on the summarized features.

This way of deep fusion provides more flexibility to the agent in approximating optimal action-values, making it possible to capture the complicated and diverse trade-offs between personal preferences and social influence for all involved users in real-world scenarios. Thus, SADQN++ is able to learn optimal policies in a more effective way. More formally, the action-value $Q(s_t, a)$ estimated by SADQN++ is given by:

$$\begin{aligned} \mathbf{h}_1^P &= \text{CONCAT}(U_i^t, V_a), \\ \mathbf{h}_2^P &= \text{ReLU}(\mathbf{W}_2^P \cdot \mathbf{h}_1^P + \mathbf{b}_2^P), \\ \mathbf{h}_3^P &= \text{ReLU}(\mathbf{W}_3^P \cdot \mathbf{h}_2^P + \mathbf{b}_3^P); \end{aligned} \tag{14}$$

$$\begin{aligned} \mathbf{h}_1^S &= \text{CONCAT}(C_i^t, V_a), \\ \mathbf{h}_2^S &= \text{ReLU}(\mathbf{W}_2^S \cdot \mathbf{h}_1^S + \mathbf{b}_2^S), \\ \mathbf{h}_3^S &= \text{ReLU}(\mathbf{W}_3^S \cdot \mathbf{h}_2^S + \mathbf{b}_3^S); \end{aligned} \tag{15}$$

$$\begin{aligned} \mathbf{h}_1^F &= \text{CONCAT}(\mathbf{h}_3^P, \mathbf{h}_3^S), \\ \mathbf{h}_2^F &= \text{ReLU}(\mathbf{W}_2^F \cdot \mathbf{h}_1^F + \mathbf{b}_2^F), \\ Q(s_t, a) &= (\mathbf{w}_3^F)^{\text{T}} \cdot \mathbf{h}_2^F + \text{b}_3^F. \end{aligned} \tag{16}$$

Here, $\mathbf{h}_l^P$, $\mathbf{W}_l^P$ and $\mathbf{b}_l^P$ denote the outputs, trainable weights and biases of l-th layer of MLP^P, respectively. Similar notations are used for MLP^S and MLP^F.

3.4 Enhancing SADQNs with Social MF-based States and Actions

To further model social information into the policy learning process, we propose an enhanced representation method that utilizes a Social MF model, instead of the standard MF model, to train the user/item embeddings (i.e., the high-level state/action representations), which can further improve the performance of SADQNs. We refer to the enhanced SADQNs as eSADQNs.

More specifically, we employ the SoRec model [16] with uniform negative sampling [25] to train three feature matrices: the user feature matrix $U \in \mathbb{R}^{f \times m}$, the item feature matrix $V \in \mathbb{R}^{f \times n}$, and the social feature matrix $W \in \mathbb{R}^{f \times m}$, by jointly factorizing the rating matrix R and the social network S in a shared latent feature space. Formally, the joint loss function to be minimized is:

$$\begin{aligned} \mathcal{L}(U, V, W) = &\sum\nolimits_{i,a} (U_i^\top V_a - R_{ia})^2 + \sum\nolimits_{i,j} (U_i^\top W_j - S_{ij})^2 \\ &+ \lambda(\|U\|_{\text{F}}^2 + \|V\|_{\text{F}}^2 + \|W\|_{\text{F}}^2), \end{aligned} \tag{17}$$

where R_{ia} is either an observed positive feedback ($R_{ia} = 1$) or a uniformly sampled negative one ($R_{ia} = 0$), S_{ij} is an observed social relation, $\|\cdot\|_{\text{F}}$ denotes the Frobenius norm, and λ is the regularization parameter.

During the user-agent interactive recommendation process, the real-time user vector U_i^t is initialized as the trained U_i at time step $t = 0$, and is always updated by performing stochastic gradient descent (SGD) based on a uniformly sampled trust relation S_{ij} and the real-time feedback R_{ia_t} received after each time step t:

$$\begin{aligned} U_i^{t+1} \leftarrow U_i^t - \alpha \big[&\Big((U_i^t)^\top V_{a_t} - R_{ia_t} \Big) V_{a_t} \\ &+ \Big((U_i^t)^\top W_j - S_{ij} \Big) W_j + \lambda U_i^t \ \big], \end{aligned} \tag{18}$$

where α is the learning rate.

3.5 Training Algorithm

To train SADQNs (i.e., the proposed Q-networks), we employ the popular Q-leaning algorithm [23]. We do not adopt the training techniques of *experience replay* and *target network* used by the original DQN [7], as they are not able to improve the Q-learning performance for our task. To make the Q-network Q converge well, sufficient transitions (s, a, r, s') of all possible states and actions are needed for Q-learning updates [6]. 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 i from training set $\mathcal{U}_{train}$ as the current target user, which will interact with the agent and 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 SADQNs is presented in Algorithm 1. The training process could last for any number of episodes as long as the Q-network has not converged. At the testing stage, the trained agent can be used to make real-time interactive recommendations for any new user j. It only needs to interact with user j, observe state s_t (U_j^t), and recommend the greedy item $a_t = \arg\max_a Q(s_t, a)$ at each time step t.

3.5.1 *Computational Complexity Analysis*

In the inner for-loop in Algorithm 1, the computation time is mainly taken in computing Q-values for available items (line 6), updating user vector U_i^{t+1} (line 8), and updating Q-network (line 10). The cost of computing Q-values is $O(n|\theta|)$, where $|\theta|$ is the number of Q-network weights and n is the number of all items. The cost of updating Q-network is $O(|\theta|)$. The cost of updating U_i^{t+1} is $O(f)$, where f is

```
Algorithm 1: Training SADQNs
   Input: $\mathcal{U}_{train}$, $R$, the trained feature matrices $U, V$
   Output: the trained Q-network $Q$
 1 Initialize $Q$ with random weights
 2 for $episode = 1, N$ do
 3     Uniformly sample a target user $i$ from $\mathcal{U}_{train}$
 4     Set initial state $s_0 = U_i^0 = U_i$
 5     for $t = 0, T-1$ do
 6         Choose the $\epsilon$-greedy item $a_t$ w.r.t. $Q(s_t, a)$
 7         Present $a_t$ to user $i$ and receive feedback $R_{ia_t}$
 8         Get $U_i^{t+1}$ according to Equation 6
 9         Set reward $r_{t+1} = R_{ia_t}$ and state $s_{t+1} = U_i^{t+1}$
10         Update $Q$'s weights on $(s_t, a_t, r_{t+1}, s_{t+1})$
             according to Equation 5
11     end
12 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
density of feedbacks	1.35%	0.18%	0.08%
#observed social relations	25,174	110,715	372,205
density of relations	2.03%	0.28%	0.09%

the dimensionality of latent feature space. Therefore, the time complexity of the training algorithm of SADQNs is $O(NT(n|\theta| + f))$, where N is the number of episodes and T is the number of time steps. Similarly, we can derive that the cost of performing T-step interactive recommendations for a new user is $O(T(n|\theta| + f))$.

4 Experiments

To validate the performance of the proposed SADQNs, we conduct extensive experiments on real-world datasets. In this section, we first introduce our experimental setup, followed by presenting the experimental results and analysis.

4.1 Experimental Setup

4.1.1 Datasets

We employ three publicly available datasets: LastFM[2] [27], Ciao[3] [28], and Epinions[4] [29] 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 that there is enough data for training and testing. The basic statistics of the obtained datasets are shown in Table 1.

4.1.2 Evaluation Methodology

To conduct experiments on interactive recommendations, we assume that the observed feedbacks in the datasets are unbiased, as proposed in [30], [31]. Similar to [32], we randomly choose 1000 unobserved (i, a) pairs of user i as the negative feedbacks. During the T-step interactive recommendation process, 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 metric indicates the ratio of positive items among the T recommended items, which is defined as: HR $= \frac{1}{T}\sum_{t=0}^{T-1} R_{ia_t}$, where a_t is the item recommended at time step t, and $R_{ia_t} = 1$ ($R_{ia_t} = 0$) if a_t is a positive (negative) item w.r.t. user i. The NDCG metric is computed by following procedure. At each time step t, a ranking list of the available items is produced according to the agent's predictions (e.g., Q-values). The DCG(t) value is calculated by: DCG$(t) = \sum_{j=1}^{k} \frac{2^{R(j)}-1}{\log_2(1+j)}$, where k is the length of the ranking list, j denotes the rank position in the ranking list, and $R(j)$ is the ground-truth feedback of the j-th item. The NDCG(t) is calculated by: NDCG$(t) = \frac{\text{DCG}(t)}{Z}$, where Z is the DCG(t) value of the optimal ranking list sorted by ground-truth feedbacks. The final NDCG is obtained by averaging the NDCG(t) values for time steps $t = 0, ..., T-1$. We truncate the ranking list at 10 to compute the NDCG@10 values, and set $T = 20$ for evaluation.

We conduct experiments for two different recommendation scenarios: cold-start and warm-start. In the *cold-start setting*, we assume that the agent has no feedback data of target user at time step $t = 0$, i.e., at the beginning of the interactive recommendation process. We randomly choose 10% users who have at least 20 positive feedbacks as the testing set $\mathcal{U}_{test}$, and others as the training set $\mathcal{U}_{train} = \mathcal{U} \backslash \mathcal{U}_{test}$, which are used to test and train the agent, respectively. The data of $\mathcal{U}_{train}$ is also used to train the feature matrices U and V. To fit the cold-start scenario, in each episode of training phase, the target user's vector U_i^0 (i.e., initial state s_0) is set to a randomized vector rather than the trained U_i (see line 4 in Algorithm 1).

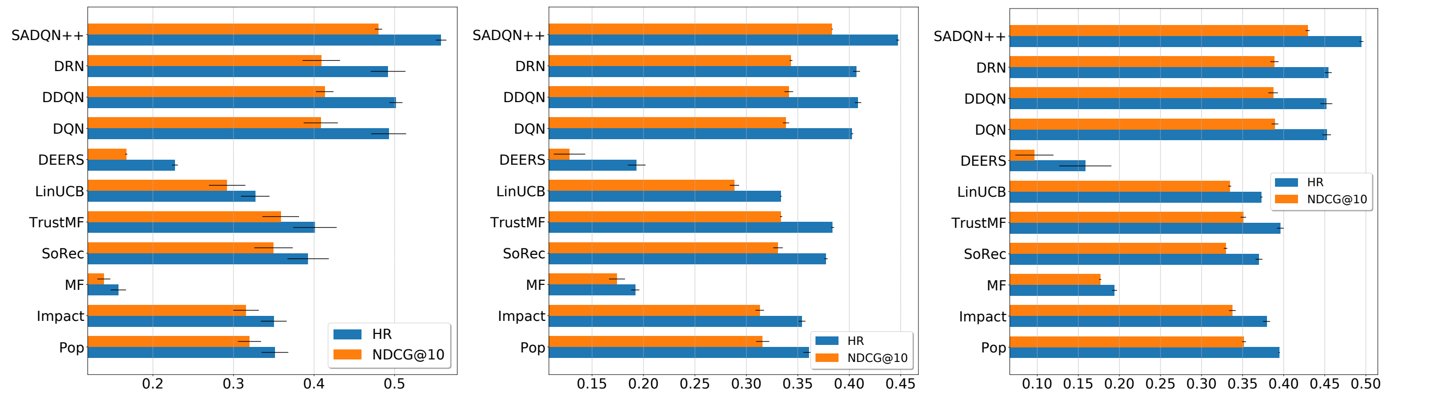
In the *warm-start setting*, we assume that the agent already has 10 positive feedbacks of target user at time step $t = 0$. We select the users who have at least 30 positive feedbacks as the target set $\mathcal{U}_{tar}$, and others as the pre-training set $\mathcal{U}_{pre} = \mathcal{U} \setminus \mathcal{U}_{tar}$. We randomly choose 10% users from $\mathcal{U}_{tar}$ as the testing set $\mathcal{U}_{test}$, and the remaining as the training set $\mathcal{U}_{train} = \mathcal{U}_{tar} \setminus \mathcal{U}_{test}$. We use R_{pre}, R_{train} and R_{test} to denote the data of $\mathcal{U}_{pre}$, $\mathcal{U}_{train}$ and $\mathcal{U}_{test}$, respectively. Then, for each user in $\mathcal{U}_{train}$ ($\mathcal{U}_{test}$), we randomly choose 10 positive feedbacks and move them from R_{train} (R_{test}) to R_{pre}. The final data R_{pre}, R_{train} and R_{test} is used to train the feature matrices, train and test the agent, respectively. In this warm-start scenario, the target user's vector U_i^0 (state s_0) already captures some preference information of him/her.

For both cold-start and warm-start settings, the testing HR or NDCG@10 value for an ideal agent will be 1. Besides, for each setting and each dataset, we conduct each experiment on 5 data splits obtained with different random seeds, and calculate the mean and standard deviation of the 5 groups of results for evaluation.

4.1.3 Baselines

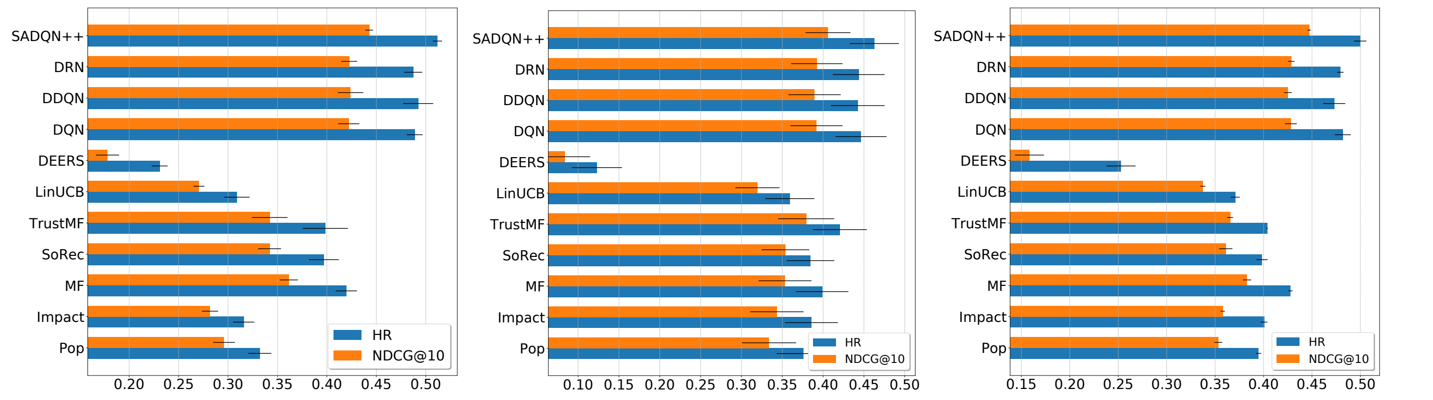
We comparatively evaluate the proposed **SADQNs** against a variety of baselines, which are listed below.

2. https://grouplens.org/datasets/hetrec-2011/
3. https://www.cse.msu.edu/~tangjili/trust.html
4. http://www.trustlet.org/downloaded_epinions.html



(a) Cold-start, LastFM (b) Cold-start, Ciao (c) Cold-start, Epinions

Fig. 4. Performance comparison against baselines on **cold-start** recommendation. The mean (bar) and standard deviation (line) of HR and NDCG@10 metrics on all three datasets are shown. The proposed SADQN++ model shows the best performance in all cases.



(a) Warm-start, LastFM (b) Warm-start, Ciao (c) Warm-start, Epinions

Fig. 5. Performance comparison against baselines on **warm-start** recommendation. The mean (bar) and standard deviation (line) of HR and NDCG@10 metrics on all three datasets are shown. The proposed SADQN++ model also shows the best performance in all cases.

1) **DQN** [7], a state-of-the-art deep reinforcement learning agent, which is originally designed for playing Atari.
2) **DDQN** [8], a state-of-the-art deep reinforcement learning agent, which extends DQN with double Q-learning [33].
3) **DRN** [10], a state-of-the-art deep reinforcement learning agent for news recommendation, which is based on Dueling DQN [34] that estimates the action-values via both value function and advantage function.
4) **DEERS** [11], a state-of-the-art deep reinforcement learning agent for recommendation, which utilizes Gated Recurrent Units (GRU) to learn state features from both positive and negative item click sequences.
5) **LinUCB** [30], a representative contextual bandit algorithm for news recommendation.
6) **SoRec** [16], a representative social matrix factorization method, which factorizes both feedback matrix and social network simultaneously.
7) **TrustMF** [21], a representative social matrix factorization method, which models the mutual influence between trusters and trustees in the trust network.
8) **MF** [24], a conventional matrix factorization model, which only exploits the feedback matrix.
9) **Impact** [35], an active learning method, which picks the item that has highest impact on other items, where the impact is computed on the user-item bipartite graph.
10) **Pop**, a popularity-based method, which picks the item which has most positive feedbacks given by users.

To make the baselines applicable to our task, we adopt the same state/action features and training scheme of SADQNs for DQN, DDQN, DRN and LinUCB, and use a negative sampling technique of uniform distribution for MF, SoRec and TrustMF. 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 activation, for DQN, DDQN and both the value and advantage functions of DRN, which lead to better performance. For DEERS, we adopt the same architecture suggested in the original reference [11]. Moreover, to make a fair comparison, we set the feature dimensionality $f = 64$ for all methods (excluding Pop, Impact and DEERS). Other parameters are tuned based on cross-validation, which are set as follows: the regularization parameter $\lambda = 0.01$, the learning rate for updating feature vectors $\alpha = 0.01$, the learning rate for updating Q-networks $\alpha = 0.0001$, the discount factor $\gamma = 0.5$, and the ϵ-greedy parameter $\epsilon = 0.1$.

It is important to notice that, our interactive recommendation task is distinctly different from session-based recommendation [36] or temporal social recommendation [37]. In their works, the recommender is developed to passively learn a predictive model from time series data such as $a_1, a_2, ..., a_{t-1}$, and to predict the next item a_t that may appear in the series. In contrast, our reinforcement learning agent is designed to actively learn a recommendation policy from user-agent interactions, and to provide items that may optimize the cumulative reward in a T-step recommendation process. As such, those Recurrent Neural Network based models [36], [37] are inapplicable to our task, and are not compared in our experiments.

Moreover, although there are many social recommendation models proposed very recently, they cannot be applied to interactive recommendation unless making critical extensions to them. Unfortunately, most of them cannot be easily extended, such as the ones proposed in [38], [39], [40], [41], [42]. Thus, we only compare with two representative social recommendation models, SoRec and TrustMF, which are very flexible and can be extended to fit our task by online learning and negative sampling.

4.2 Performance Comparison against Baselines

We now compare the performance of SADQN++ (the best performing variant of SADQN) against the baseline methods. Figure 4 and 5 show the comparison results in terms of the mean (bar) and standard deviation (line) of HR and NDCG@10 metrics for cold-start and warm-start recommendations, respectively. From these results, we have the following main findings.

For cold-start recommendation (see Figure 4), the proposed SADQN++ model shows the best performance in terms of both metrics on all datasets. It remarkably outperforms the four deep reinforcement learning agents DQN, DDQN, DRN and DEERS that fail to consider social influence, as well as other types of baselines. For example, its improvements in HR metric against the best performing baseline are 11.19%, 9.47% and 8.88% on LastFM, Ciao and Epinions datasets, respectively. These results not only verify the capability of SADQN++, but also demonstrate that social influence plays a fundamental role in improving the performance of deep reinforcement learning recommenders.

The DQN, DDQN and DRN agents show the second-class performance, while DEERS performs almost the worst which implies that it might be inappropriate to our task. The traditional matrix factorization model MF shows poor performance, as no feedback data is available at time step $t = 0$, while the social recommendation models SoRec and TrustMF demonstrate much better performance. Besides, the popularity method Pop and active learning method Impact are also competitive baselines in the cold-start setting.

For warm-start recommendation (see Figure 5), the proposed SADQN++ model also shows significantly better performance than the competitors. Specifically, its improvements in HR (NDCG@10) metric against the best performing baseline are 3.9%, 3.6% and 3.7% (4.4%, 3.5% and 4.3%) on LastFM, Ciao and Epinions datasets, respectively. The results of baselines show similar trends compared to cold-start setting, with one exception of the MF model, which shows competitive performance in the warm-start setting.

4.3 The Impact of Social Influence

To quantitatively analyze the impact of social influence, here, we make a more detailed comparison among the four variants of SADQN. They are the pure personal model SADQN^P (which is equivalent to the DQN baseline), the pure social model SADQN^S, the linear fusion model SADQN and the deep fusion model SADQN++, respectively (see Section 3 for more details). The comparison results of the mean and standard deviation of HR and NDCG@10 metrics are reported in Table 2. The relative improvements of SADQN^S, SADQN and SADQN++ against SADQN^P are shown in the brackets, which tell us clearly how social influence increasingly improves the performance when we model it from shallowly to deeply. Also, the best result in each case is highlighted. From this part of results, we observe the following main points.

The deep fusion model SADQN++ performs much better than others. In particular, its relative improvements against the personal model SADQN^P are at least larger than 9.0% for cold-start recommendation, and 3.5% for warm-start recommendation. Also, the improvements in cold-start setting show an interesting trend that they are totally consistent with the densities of social relations in the datasets. More specifically, the order of the improvements in HR (NDCG@10) metric on LastFM, Ciao and Epinions datasets is $13.2\% > 11.0\% > 9.3\%$ ($17.5\% > 13.2\% > 10.2\%$), same as the order of relation densities $2.03\% > 0.28\% > 0.09\%$ (see Table 1). This demonstrates that, more social relations SADQN++ exploits, more benefits it will obtain.

The linear fusion model SADQN performs second-best in all cases, and shows similar trends with SADQN++.

The pure social model SADQN^S also performs better than SADQN^P in cold-start setting (except for the case of HR metric on Epinions dataset), but shows worse performance in warm-start setting (in most cases). This implies that, when the social network data is extremely sparse, or when the user-item feedback data is sufficient, the SADQN^S model purely using social influence cannot achieve desirable recommendation performance.

4.4 The Impact of Social MF-based States/Actions

To investigate the impact of the proposed social MF-based states/actions, we compare the performance of SADQN++ with its enhanced version, eSADQN++. This experiment is only conducted in cold-start recommendation setting. The comparison results in terms of both HR and NDCG@10 metrics on all three datasets are shown in Table 3, where the bold font indicates the best result in each case. As we can see, eSADQN++ consistently performs better than SADQN++ with significant margins. This demonstrate that by additionally modeling social information into the latent state/action representations, our proposed SADQNs can be further improved.

4.5 Comparison of Different Attention Mechanisms

We now compare the performance of different attention mechanisms discussed in Section 3.2. We conduct an experiment to compare three SADQN agents which adopt the attention mechanisms GAT [26], DOT (Equation 10), and CONCAT (Equation 7, i.e., the default one used by SADQN), respectively. This experiment is only conducted in cold-start recommendation setting. The comparison results in terms of both HR and NDCG@10 metrics on all three datasets are shown in Table 4, where the bold font indicates the best result in each case. The two attention mechanisms CONCAT and DOT perform very closely, and both outperform GAT.

TABLE 2
The Experimental Results of Different Variants of SADQN

Dataset	Model	Cold-start Recommendation		Warm-start Recommendation	
		HR	NDCG@10	HR	NDCG@10
LastFM	SADQN^P	0.4927±0.0216	0.4083±0.0212	0.4888±0.0080	0.4220±0.0108
	SADQN^S	0.5236±0.0123 (+6.2%)	0.4644±0.0136 (+13.7%)	0.4768±0.0097 (-2.4%)	0.4131±0.0065 (-2.1%)
	SADQN	0.5435±0.0081 (+10.3%)	0.4724±0.0096 (+15.7%)	0.4970±0.0132 (+1.6%)	0.4297±0.0060 (+1.8%)
	SADQN++	**0.5577±0.0065 (+13.2%)**	**0.4800±0.0045 (+17.5%)**	**0.5118±0.0047 (+4.7%)**	**0.4427±0.0039 (+4.9%)**
Ciao	SADQN^P	0.4028±0.0010	0.3386±0.0030	0.4464±0.0313	0.3918±0.0319
	SADQN^S	0.4174±0.0054 (+3.6%)	0.3627±0.0040 (+7.1%)	0.4474±0.0347 (+0.2%)	0.3916±0.0305 (+0.0%)
	SADQN	0.4247±0.0077 (+5.4%)	0.3644±0.0016 (+7.6%)	0.4512±0.0301 (+1.1%)	0.3970±0.0273 (+1.3%)
	SADQN++	**0.4472±0.0014 (+11.0%)**	**0.3834±0.0006 (+13.2%)**	**0.4628±0.0301 (+3.6%)**	**0.4060±0.0274 (+3.6%)**
Epinions	SADQN^P	0.4522±0.0053	0.3894±0.0041	0.4819±0.0082	0.4282±0.0062
	SADQN^S	0.4465±0.0005 (-1.2%)	0.3971±0.0016 (+2.0%)	0.4684±0.0048 (-2.8%)	0.4163±0.0040 (-2.8%)
	SADQN	0.4782±0.0017 (+5.7%)	0.4154±0.0018 (+6.7%)	0.4877±0.0014 (+1.2%)	0.4353±0.0007 (+1.6%)
	SADQN++	**0.4945±0.0024 (+9.3%)**	**0.4291±0.0025 (+10.2%)**	**0.4998±0.0064 (+3.7%)**	**0.4471±0.0017 (+4.4%)**

TABLE 3
Comparison between SADQN++ and eSADQN++

Dataset	Model	HR	NDCG@10
LastFM	SADQN++	0.5577±0.0065	0.4800±0.0045
	eSADQN++	**0.5697±0.0059**	**0.4966±0.0094**
Ciao	SADQN++	0.4472±0.0014	0.3834±0.0006
	eSADQN++	**0.4556±0.0039**	**0.3881±0.0035**
Epinions	SADQN++	0.4945±0.0024	0.4291±0.0025
	eSADQN++	**0.5066±0.0054**	**0.4447±0.0038**

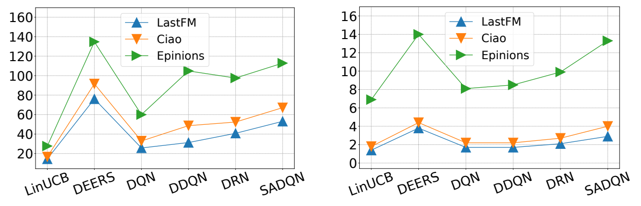
TABLE 4
The Results of SADQN with Different Attention Mechanisms

Dataset	Attention	HR	NDCG@10
LastFM	GAT	0.5191±0.0114	0.4393±0.0127
	DOT	**0.5438±0.0036**	0.4717±0.0084
	CONCAT	0.5435±0.0076	**0.4724±0.0091**
Ciao	GAT	0.4179±0.0060	0.3567±0.0082
	DOT	**0.4256±0.0031**	**0.3670±0.0014**
	CONCAT	0.4247±0.0072	0.3644±0.0011
Epinions	GAT	0.4667±0.0018	0.4033±0.0005
	DOT	0.4755±0.0016	0.4142±0.0030
	CONCAT	**0.4782±0.0012**	**0.4154±0.0013**

4.6 Run-time Analysis

As analyzed in Section 3.5.1, the time complexity of the training algorithm of SADQNs is $O(NT(n|\theta| + f))$, where N is the number of episodes for training, T is the number of time steps in each episode, n is number of items in the dataset, $|\theta|$ is the number of Q-network weights, and f is the dimensionality of latent feature space. Since f is usually a small constant in practice (f=64 in our experiments), we can rewrite the time complexity as $O(Ln|\theta|)$, where $L = NT$ denotes the number of Q-learning updates that is needed to make the Q-network converge. In other words, the time cost for training SADQN agent is related to three basic variables: the size of Q-network, the size of data, and the convergence of Q-learning algorithm. Similarly, we can derive that the time cost for testing SADQN agent is related to the size of Q-network and the size of data.

Here, we show the run-time of SADQN for cold-start recommendation on a single-GPU machine. The time cost for warm-start recommendation would be similar. We also show the time costs of RL-based baselines for comparison. The non-RL methods are not compared here, as their time costs are much lower and can be ignored in comparison to RL-based methods. Both the training time (in minutes) and the testing time (in seconds) of one run of the compared methods on all three datasets are shown in Figure 6. Overall, the testing costs of all compared methods are quite low, which demonstrates that they are able to perform real-time online recommendations. For the training cost, the proposed SADQN is lower than DEERS, close to DDQN and DRN, but higher than DQN and LinUCB. The results demonstrate that our approach is able to improve the performance of the state-of-the-art RL agents, with acceptable additional cost.



(a) Training time (in minutes) (b) Testing time (in seconds)

Fig. 6. Comparison of the run-time of RL-based methods.

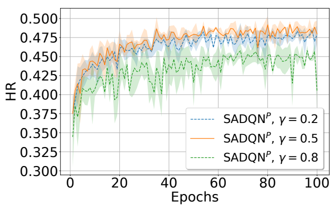
4.7 Parameter Analysis

In this subsection, we conduct experiments to show how the proposed SADQN agents behave with different settings of some important hyperparameters. When analyzing a specific parameter, others are fixed to the default settings. More specifically, we use the simplest variant of SADQN, SADQN^P, as an example to conduct the analysis. In addition, these experiments are only conducted in cold-start setting using LastFM dataset.

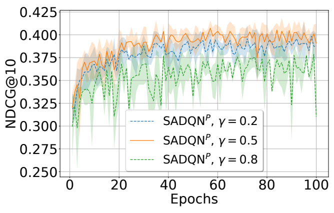
4.7.1 *Effect of the Discount Factor* γ

The discount factor $\gamma \in [0, 1]$ balances the trade-off between future rewards and immediate rewards when estimating Q-values. In general, a higher γ will produce a more farsighted policy than a lower one. However, the selection of the optimal value of γ usually depends on the natures of specific RL tasks. This is still an open problem in the literature. More discussions can be found in [6].

Here, we vary γ in $\{0.2, 0.5, 0.8\}$ to compare the performance of SADQN^P. The learning curves in terms of both mean (line) and standard deviation (shadow) of HR and NDCG@10 metrics are shown in Figure 7, where each epoch corresponds to 50k Q-learning updates in Algorithm 1. Note that the learning curves are plotted from epoch 1. The results demonstrate two main points. First, SADQN^P with $\gamma = 0.5$ achieves the best performance. Second, SADQN^P with a lower γ learns more stably than a higher one.

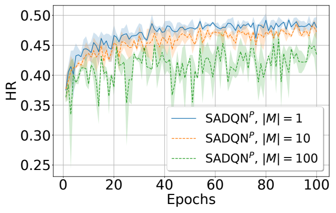


(a) HR

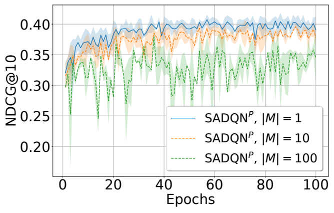


(b) NDCG@10

Fig. 7. Effect of the discount factor γ on the performance of SADQN^P.



(a) HR



(b) NDCG@10

Fig. 8. Effect of the experience replay on the performance of SADQN^P.

In our interactive recommendation task, the $\gamma = 0.5$ is preferred, as shown in the results. We notice that our results show a somewhat different trend with the results in many traditional RL tasks such as Atari games (where a higher γ is preferred). The possible reason is that the immediate reward of a recommended item is relatively more important than the future rewards affected by this item. This makes sense in interactive recommendation since the agent needs to recommend interesting items (with high immediate rewards) as soon as possible during interactions, especially in our cold-start setting. Moreover, unlike most of traditional tasks, the rewards in our recommendation task are not sparse (at each time step, the agent will receive a feedback as reward).

4.7.2 *Effect of the Experience Replay*

The training trick of experience replay was first proposed by [43]. To perform experience replay, a memory M is needed to store the transitions that the agent collects during the user-agent interactive process. At each time step, the Q-network is updated based on a minibatch of transitions sampled from M, rather than based on the currently observed transition (see line 10 in Algorithm 1). The experience replay has shown appealing performance improvements in some reinforcement learning tasks such as playing Atari [7].

Here, we validate whether and how much the experience replay will benefit SADQN^P for a totally different recommendation task. To do this, we vary the memory size $|M| \in \{1, 10, 100\}$ to check how the agent's performance changes. Note that $|M| = 1$ (i.e., the default setting used by SADQNs) implies that the agent actually does not adopt experience replay. The comparison results are shown in Figure 8, which clearly tell us the same trend: SADQN^P with $|M| = 1$ achieves the best performance, and its performance will be consistently reduced when the memory becomes larger. The implies, different from traditional tasks, the use of experience replay will significantly reduce the agent's performance for our recommendation task.

5 Related Work

In this section, we review the most relevant studies to our work, which can be categorized into two classes: (generalized) collaborative filtering and reinforcement learning.

5.1 Collaborative Filtering

Collaborative filtering (CF) is probably the most prevalent and successful approach to building personalized recommender systems. The core idea behind CF is to infer users' personal preferences from their nearest neighbors (i.e., collaborators), and thus make personalized and collaborative recommendations for all users.

Early CF methods are essentially some memory-based (or heuristic-based) algorithms, which aim to predict the rating of a given user-item pair based on the aggregation of the ratings of other similar users (user-based CF), similar items (item-based CF), or both of them (Hybrid CF) [2], [44], [45], [46], [47], [48]. These memory-based CF methods have to compute the similarities of users or items over the entire user-item matrix, which usually have limited prediction accuracy and cannot be applied to large datasets. To improve the prediction accuracy and scalability, researchers incorporate the techniques in machine learning such as matrix factorization and neural networks, and design a large number of model-based CF methods that aim to learn an accurate prediction model from data. On the other hand, to alleviate the issues of cold-start and data sparsity, many researchers exploit additional data sources of the available social networks to help infer users' preferences and develop a number of social CF methods. We will discuss these methods in the following.

5.1.1 *Matrix Factorization*

Matrix factorization (MF) maps users and items into a latent feature space and makes predictions via the inner product between the feature vectors of users and items. A lot of MF models have been proposed for the rating prediction task in explicit feedback datasets [24], [49], [50], [51]. A representative MF model is the probabilistic matrix factorization (PMF) proposed by Salakhutdinov and Mnih [50], which alleviates the over-fitting problem by defining Gaussian priors on the latent feature vectors. Koren [51] propose another famous MF model, SVD++, which achieves dominant performance in the rating prediction task and win the 2008 Netflix Prize. The core prediction model of SVD++ consists of: the inner product of the user and item feature vectors, the average rating in the observed data, the user bias parameter, the item bias parameter, and the implicit feedback term.

On the other hand, a number of MF models have been proposed for the top-k item recommendation task in implicit feedback datasets [52], [53], [54]. Hu et al. [52] investigate the special properties of implicit feedback datasets in a TV show recommender system, and propose a MF model by incorporating a confidence factor of users' positive interests into the squared loss function. Pan et al. [53] improve the performance of MF for implicit feedback recommendation by proposing several negative example weighting/sampling strategies, including uniform weighting/sampling, user-oriented weighting/sampling, and item-oriented weighting/sampling.

5.1.2 *Neural Collaborative Filtering*

Some researchers generalize the idea of CF into deep learning and design a number of neural CF methods. For instance, Wang et al. [55] propose a hierarchical Bayesian model, named collaborative deep learning (CDL), which jointly performs deep representation learning for the content information and CF for the feedback matrix. Wang et al. [56] develop a collaborative recurrent autoencoder (CRAE) which models the generation of content sequences in the CF setting. Zhang et al. [57] propose an integrated autoencoder-based framework, termed collaborative knowledge base embedding (CKE), which jointly learns the latent representations in CF and the items' semantic representations from the knowledge base.

He et al. [58] propose a neural matrix factorization model, named NeuMF, which integrates multi-layer perceptron (MLP) with generalized matrix factorization to make predictions, and is trained by minimizing a binary cross-entropy loss over both observed positive user-item pairs and sampled negative ones. Song et al. [59] propose a similar neural network architecture to make predictions, but the network is trained with a different pairwise ranking loss function. Ebesu et al. [60] propose a collaborative memory network (CMN) method, which integrates memory networks with MF models to perform top-k recommendations. Liang et al. [61] propose a neural CF method based on variational autoencoders for implicit feedback recommendations, which utilizes a generative model with multinomial likelihood and uses Bayesian inference for parameter estimation.

Recently, a number of neural CF methods that leverage graph convolutional networks (GCNs) have been proposed for a variety of recommendation domains and have demonstrated appealing performance [62], [63], [64]. Besides, more works on neural CF methods can be found in a recent survey on deep learning based recommender systems [5].

5.1.3 *Social Collaborative Filtering*

A number of memory-based social CF methods [65], [66], [67] explore the trust propagation in social network, which generate rating predictions for a target user by directly aggregating the feedback data of his/her trusted friends. These memory-based social CF methods are able to improve the coverage of recommendations compared to traditional memory-based CF methods, but are not suitable to large-scale datasets as they need to compute similarities over the entire rating matrix and the whole social network.

Model-based social CF methods [16], [17], [18], [19], [21], [68], [69] employ the technique of MF to exploit the available social network data, which can be applied to large datasets. SoRec [16], TrustMF [21] and PSLF [19] factorize simultaneously the user-item rating matrix and the user-user social network simultaneously in a shared latent feature space. RSTE [17] fuses target user's interests and his/her trusted friends' tastes to model/predict a specific rating of a user-item pair. SocialMF [18] is similar to RSTE, but uses a different implementation to take into account the interests of trusted friends by incorporating a regularization term to control the distance between target user's feature vector and the averaged feature vector of his/her trusted friends. In [68], the authors introduce several social regularization terms that are similar to the one in SocialMF. The main difference is that the trust value in social regularization term is replaced by the Pearson correlation coefficient (PCC) calculated on users' rating data.

Recently, several complex social CF models beyond MF have been proposed for social recommender systems [38], [39], [40], [41], [42]. SREPS [38] simultaneously models the structural information in the social network, and the rating and consumption information in the user-item feedback data under an essential preference space, by using both network embedding and MF. GraphRec [41] utilizes graph neural networks to learn user and item latent feature vectors from both user-item feedback graph and user-user social graph. SamWalker [42] simultaneously learns personalized data confidence and draws informative training instances by leveraging the social network information. SAMN [39] utilizes an attention-based memory module to learn user-friend relation vectors, and builds a friend-level attention component to adaptively select informative friends for user preference modeling. DANSER [40] introduces dual graph attention networks to collaboratively learn representations for two-fold social effects, which are captured by a user-specific attention weight and a dynamic context-aware attention weight, respectively.

5.2 Reinforcement Learning

5.2.1 *Tabular Reinforcement Learning*

Shani et al. [70] propose an MDP-based approach to book recommendation, which models the user clicked item sequence as the MDP's states and solves the MDP via dynamic programming. Unfortunately, for most of real-world recommender systems, the MDP is always unknown since both the state-transition function and reward function cannot be explicitly specified. Thus, model-based RL methods such as dynamic programming are not able to provide solutions in those cases. Instead, model-free RL methods do not require the specific dynamics of the environment's model, and are able to update the agent's policy towards an optimal one by interacting with environment in a trial-and-error fashion.

Temporal-difference learning (e.g., Q-learning [23]) is one of the most popular model-free RL algorithms, which aims to estimate the optimal state-value or action-value function (corresponding to an optimal policy) by using bootstrapping [6]. Researchers have applied the temporal-difference based algorithms to some small-scale recommender systems. Taghipour et al. [71] propose a Q-learning based recommendation approach to a webpage recommender system. Silver et al. [72] propose a concurrent RL framework, which utilizes a variant of temporal-difference learning to learn policies efficiently from partial interaction sequences of users. Choi et al. [73] formulate the recommendation problem as a gridworld game by using a biclustering technique to convert the user-item matrix to a series of grid states, and estimate the optimal policy with some Q-learning based algorithms.

5.2.2 *Deep Reinforcement Learning*

In spite of their good convergence guarantees, the aforementioned tabular RL methods require maintaining a lookup table of all states or/and actions during the policy learning process. As a result, they are not able to handle the high-dimensional or continuous state and action spaces of most real-world recommender systems, and are of poor generalization abilities with respect to unseen states or actions. To overcome the weaknesses of tabular RL methods, an effective way is to leverage the technique of value function approximation to approximate the optimal action-value function (corresponding to an optimal policy) by using a parameterized function approximator [6], [43]. On the other hand, instead of using value function approximation, an alternative way is to directly parameterize the policy itself and update the parameterized policy towards an optimal one according to the policy gradient theorem [6]. Next, we first review the existing deep RL agents that utilize deep neural networks as the function approximator to approximate the optimal action-value function (value-based deep RL) or the optimal policy (policy-based deep RL), followed by discussing some supervised learning techniques in deep RL-based recommender systems.

Value-based Deep RL. Deep Q-network (DQN) [7] is probably the most popular deep RL agent, which has demonstrated human-level or even better performance in playing the Atari games. Over the past few years, a lot of extensions or modifications to DQN such as Dueling DQN [34], Double DQN [8], and Noisy DQN [74] have been proposed for game-based RL tasks and have shown better performance. Due to the successes of DQN-based agents in game-based tasks, a number of researchers propose to incorporate the techniques of DQN into recommender systems and design a few DQN-based recommendation agents in different recommendation domains [10], [11], [13]. DEERS [11] is a DQN-based agent designed for product recommendation in e-commerce sites. It first utilizes gated recurrent units as the state representation module to learn high-level state representations from both the clicked and skipped item sequences of users, and then employ a parallel action-value prediction module to estimate action-values based on the two types of high-level state representations. DRN [10] applies the Dueling DQN agent to news recommendation, which estimates action-values based on the linear combination of state-value function and advantage function. Particularly, it relies on a manual state/action representation module that incorporates various hand-crafted features of users, news and contexts into the representations of states and actions. Robust DQN [13] extends the DDQN agent to online tip recommendation by proposing stratified sampling replay strategy and approximate regretted reward.

Policy-based Deep RL. Another major type of deep RL agents are based on the policy gradient theorem, such as Monte Carlo policy gradient (a.k.a. REINFORCE [75]) and actor-critic policy gradient [6]. Chen et al. [76] propose a REINFORCE-based deep RL agent to recommendation, by incorporating off-policy correction to tackle the biases in the logged user feedback data collected from multiple behavior policies. Wang et al. [77] propose an action-critic policy gradient based deep RL agent, which utilizes recurrent neural networks to address the issue of partially observed states in real-world treatment recommender systems. Chen et al. [78] propose a tree-structured policy gradient recommendation (TPGR) framework, which builds a balanced hierarchical clustering tree over the items and formulates the item picking problem as path seeking from the root to a leaf of the tree. Moreover, deep deterministic policy gradient (DDPG) [79] is a popular actor-critic deep RL agent based on the deterministic policy gradient theorem [80], which is able to hand continuous action space and learn deterministic policies. A number of DDPG-based recommendation agents have been proposed to a variety of recommendation domains [12], [81], [82], [83].

Supervised Learning Techniques in Deep RL. A number of studies incorporate the techniques of supervised learning into deep RL-based recommender systems. Chen et al. [84] train a user behavior model based on offline logged data by leveraging generative adversarial networks. The trained user behavior model is treated as an environment simulator, which is used to interact the agent and promote the learning of RL-based recommendation policies. Shi et al. [85] build a similar environment simulator with generative adversarial networks based on the logged data in a large-scale e-commerce website. Zou et al. [86] propose a Pseudo Dyna-Q framework, which iteratively learns an environment simulator and a DQN agent from the logged data. Besides, Liu et al. [87] propose a supervised learning-based user/item embedding component for deep RL-based recommendation agents, in order to learn more effective high-level state/action representations.

6 Conclusion

Deep reinforcement learning has been successfully applied to recommender systems, but still heavily suffer from data sparsity and cold-start in real-world tasks. In this work, we addressed these issues by strengthening the state-of-the-art deep reinforcement learning recommenders with social influence among users. We developed a class of Social Attentive Deep Q-networks (SADQNs) to estimate action-values based on the preferences of both individual users and social neighbors, by successfully utilizing an attention mechanism to model the social influence between them. In particular, we proposed an enhanced variant of SADQN, termed SADQN++, which is able to model the complicated trade-offs between personal preferences and social influence for all users, making the agent more powerful and flexible in learning optimal policies.

We conducted extensive experiments on three real-world datasets to verify the effectiveness and efficiency of the proposed SADQNs. The results have demonstrated that social influence plays a fundamental role in improving the recommendation performance of deep reinforcement learning, especially in the cold-start recommendation scenarios. More importantly, the significant improvements of SADQNs over the state-of-the-art agents are accomplished with reasonable computation cost.

We notice that our SADQNs might have computation issue when deploying them in production systems that contain billions of items. This is because during prediction, all candidate tiems need to go through the neural network once, meaning that N feedforward passes are needed for N candidate items. This issue is also faced by some deep learning based methods such as NCF [58], but is not faced by other methods such as CDL [55] since it adopts the autoencoder architecture that will take the whole candidate items as input. To address the computation issue of our methods, a possible approach is to employ another type of DQN architecture that only takes state as input and outputs the action-values of all candidate items. This new architecture only needs one feedforward pass for N candidate items (additional computation cost only appears in the last layer of the Q-network). However, this new architecture might reduce the recommendation quality, as it ignores the information of item representations. We will explore more about the new architecture in the future work.

References

[1] N. Rubens, M. Elahi, M. Sugiyama, and D. Kaplan, "Active learning in recommender systems," in *Recommender systems handbook*. Springer, 2015, pp. 809–846.

[2] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *TKDE*, vol. 17, no. 6, pp. 734–749, 2005.

[3] F. Ricci, L. Rokach, and B. Shapira, "Introduction to recommender systems handbook," in *Recommender systems handbook*. Springer, 2011, pp. 1–35.

[4] Y. Koren and R. Bell, "Advances in collaborative filtering," in *Recommender systems handbook*. Springer, 2015, pp. 77–118.

[5] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *CSUR*, vol. 52, no. 1, p. 5, 2019.

[6] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[8] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *AAAI*, 2016.

[9] 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.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[10] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li, "Drn: A deep reinforcement learning framework for news recommendation," in *WWW*. IW3C2, 2018, pp. 167–176.

[11] X. Zhao, L. Zhang, Z. Ding, L. Xia, J. Tang, and D. Yin, "Recommendations with negative feedback via pairwise deep reinforcement learning," in *SIGKDD*. ACM, 2018, pp. 1040–1048.

[12] X. Zhao, L. Xia, L. Zhang, Z. Ding, D. Yin, and J. Tang, "Deep reinforcement learning for page-wise recommendations," in *RecSys*. ACM, 2018, pp. 95–103.

[13] S. Chen, Y. Yu, Q. Da, J. Tan, H. Huang, and H. Tang, "Stabilizing reinforcement learning in dynamic environment with application to online recommendation," in *SIGKDD*. ACM, 2018, pp. 1187–1196.

[14] R. M. Bond, C. J. Fariss, J. J. Jones, A. D. Kramer, C. Marlow, J. E. Settle, and J. H. Fowler, "A 61-million-person experiment in social influence and political mobilization," *Nature*, vol. 489, no. 7415, p. 295, 2012.

[15] E. Bakshy, I. Rosenn, C. Marlow, and L. Adamic, "The role of social networks in information diffusion," in *WWW*. ACM, 2012, pp. 519–528.

[16] H. Ma, H. Yang, M. R. Lyu, and I. King, "Sorec: social recommendation using probabilistic matrix factorization," in *CIKM*. ACM, 2008, pp. 931–940.

[17] H. Ma, I. King, and M. R. Lyu, "Learning to recommend with social trust ensemble," in *SIGIR*. ACM, 2009, pp. 203–210.

[18] M. Jamali and M. Ester, "A matrix factorization technique with trust propagation for recommendation in social networks," in *RecSys*. ACM, 2010, pp. 135–142.

[19] Y. Shen and R. Jin, "Learning personal+ social latent factor model for social recommendation," in *SIGKDD*. ACM, 2012, pp. 1303–1311.

[20] J. Tang, X. Hu, and H. Liu, "Social recommendation: a review," *Social Network Analysis and Mining*, vol. 3, no. 4, pp. 1113–1133, 2013.

[21] B. Yang, Y. Lei, J. Liu, and W. Li, "Social collaborative filtering by trust," *TPAMI*, vol. 39, no. 8, pp. 1633–1647, 2017.

[22] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[23] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[24] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[25] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *UAI*, 2009, pp. 452–461.

[26] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv*, 2017.

[27] I. Cantador, P. L. Brusilovsky, and T. Kuflik, *HetRec2011*. ACM, 2011.

[28] J. Tang, H. Gao, H. Liu, and A. Das Sarma, "etrust: Understanding trust evolution in an online world," in *SIGKDD*. ACM, 2012, pp. 253–261.

[29] P. Massa and P. Avesani, "Trust-aware recommender systems," in *RecSys*. ACM, 2007, pp. 17–24.

[30] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *WWW*. ACM, 2010, pp. 661–670.

[31] X. Zhao, W. Zhang, and J. Wang, "Interactive collaborative filtering," in *CIKM*. ACM, 2013, pp. 1411–1420.

[32] P. Cremonesi, Y. Koren, and R. Turrin, "Performance of recommender algorithms on top-n recommendation tasks," in *RecSys*. ACM, 2010, pp. 39–46.

[33] H. V. Hasselt, "Double q-learning," in *NIPS*, 2010, pp. 2613–2621.

[34] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[35] C. E. Mello, M. A. Aufaure, and G. Zimbrao, "Active learning driven by rating impact analysis," in *RecSys*. ACM, 2010, pp. 341–344.

[36] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," in *ICLR*, 2016.

[37] P. Sun, L. Wu, and M. Wang, "Attentive recurrent social recommendation," in *SIGIR*. ACM, 2018, pp. 185–194.

[38] C. Liu, C. Zhou, J. Wu, Y. Hu, and L. Guo, "Social recommendation with an essential preference space." in *AAAI*, 2018.

[39] C. Chen, M. Zhang, Y. Liu, and S. Ma, "Social attentional memory network: Modeling aspect-and friend-level differences in recommendation," in *WSDM*. ACM, 2019, pp. 177–185.

[40] Q. Wu, H. Zhang, X. Gao, P. He, P. Weng, H. Gao, and G. Chen, "Dual graph attention networks for deep latent representation of multifaceted social effects in recommender systems," pp. 2091–2102, 2019.

[41] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," pp. 417–426, 2019.

[42] J. Chen, C. Wang, S. Zhou, Q. Shi, Y. Feng, and C. Chen, "Samwalker: Social recommendation with informative sampling strategy," in *WWW*. ACM, 2019, pp. 228–239.

[43] L. Lin, "Reinforcement learning for robots using neural networks," CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, Tech. Rep., 1993.

[44] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Communications of the ACM*, vol. 35, no. 12, pp. 61–70, 1992.

[45] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: an open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM, 1994, pp. 175–186.

[46] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *WWW*. ACM, 2001, pp. 285–295.

[47] M. Deshpande and G. Karypis, "Item-based top-n recommendation algorithms," *TOIS*, vol. 22, no. 1, pp. 143–177, 2004.

[48] 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.

[49] J. D. Rennie and N. Srebro, "Fast maximum margin matrix factorization for collaborative prediction," in *ICML*. ACM, 2005, pp. 713–719.

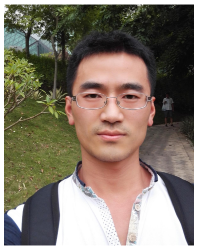
[50] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *NIPS*, 2007, pp. 1257–1264.

[51] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *SIGKDD*. ACM, 2008, pp. 426–434.

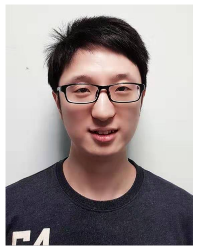
[52] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *ICDM*. IEEE, 2008, pp. 263–272.

[53] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang, "One-class collaborative filtering," in *ICDM*. IEEE, 2008, pp. 502–511.

[54] X. He, H. Zhang, M. Kan, and T. Chua, "Fast matrix factorization for online recommendation with implicit feedback," in *SIGIR*. ACM, 2016, pp. 549–558.
[55] H. Wang, N. Wang, and D. Yeung, "Collaborative deep learning for recommender systems," in *SIGKDD*. ACM, 2015, pp. 1235–1244.
[56] H. Wang, X. Shi, and D. Yeung, "Collaborative recurrent autoencoder: Recommend while learning to fill in the blanks," in *NIPS*, 2016, pp. 415–423.
[57] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W. Ma, "Collaborative knowledge base embedding for recommender systems," in *SIGKDD*, 2016, pp. 353–362.
[58] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural collaborative filtering," in *WWW*. IW3C2, 2017, pp. 173–182.
[59] B. Song, X. Yang, Y. Cao, and C. Xu, "Neural collaborative ranking," in *CIKM*. ACM, 2018, pp. 1353–1362.
[60] T. Ebesu, B. Shen, and Y. Fang, "Collaborative memory network for recommendation systems," in *SIGIR*. ACM, 2018, pp. 515–524.
[61] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, "Variational autoencoders for collaborative filtering," in *WWW*. International World Wide Web Conferences Steering Committee, 2018, pp. 689–698.
[62] R. v. d. Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *arXiv preprint arXiv:1706.02263*, 2017.
[63] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *SIGKDD*. ACM, 2018, pp. 974–983.
[64] X. Wang, X. He, M. Wang, F. Feng, and T. Chua, "Neural graph collaborative filtering," in *SIGIR*. ACM, 2019.
[65] P. Massa and P. Avesani, "Trust-aware recommender systems," in *RecSys*. ACM, 2007, pp. 17–24.
[66] M. Jamali and M. Ester, "Trustwalker: a random walk model for combining trust-based and item-based recommendation," in *SIGKDD*. ACM, 2009, pp. 397–406.
[67] P. Victor, C. Cornelis, M. De Cock, and P. P. Da Silva, "Gradual trust and distrust in recommender systems," *Fuzzy Sets and Systems*, vol. 160, no. 10, pp. 1367–1382, 2009.
[68] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King, "Recommender systems with social regularization," in *WSDM*. ACM, 2011, pp. 287–296.
[69] A. J. B. Chaney, D. M. Blei, and T. Eliassi-Rad, "A probabilistic model for using social networks in personalized item recommendation," in *RecSys*. ACM, 2015, pp. 43–50.
[70] G. Shani, D. Heckerman, and R. I. Brafman, "An mdp-based recommender system," *JMLR*, vol. 6, no. Sep, pp. 1265–1295, 2005.
[71] N. Taghipour, A. Kardan, and S. S. Ghidary, "Usage-based web recommendations: a reinforcement learning approach," in *RecSys*. ACM, 2007, pp. 113–120.
[72] D. Silver, L. Newnham, D. Barker, S. Weller, and J. McFall, "Concurrent reinforcement learning from customer interactions," in *ICML*, 2013, pp. 924–932.
[73] S. Choi, H. Ha, U. Hwang, C. Kim, J. Ha, and S. Yoon, "Reinforcement learning based recommender system using biclustering technique," *arXiv preprint arXiv:1801.05532*, 2018.
[74] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin *et al.*, "Noisy networks for exploration," in *ICLR*, 2018.
[75] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.
[76] M. Chen, A. Beutel, P. Covington, S. Jain, F. Belletti, and E. H. Chi, "Top-k off-policy correction for a reinforce recommender system," in *WSDM*. ACM, 2019, pp. 456–464.
[77] L. Wang, W. Zhang, X. He, and H. Zha, "Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation," in *SIGKDD*. ACM, 2018, pp. 2447–2456.
[78] H. Chen, X. Dai, H. Cai, W. Zhang, X. Wang, R. Tang, Y. Zhang, and Y. Yu, "Large-scale interactive recommendation with tree-structured policy gradient," in *AAAI*, vol. 33, 2019, pp. 3312–3320.
[79] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *ICLR*, 2016.
[80] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014, pp. 387–395.
[81] Y. Hu, Q. Da, A. Zeng, Y. Yu, and Y. Xu, "Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application," in *SIGKDD*. ACM, 2018, pp. 368–377.
[82] F. Liu, R. Tang, X. Li, W. Zhang, Y. Ye, H. Chen, H. Guo, and Y. Zhang, "Deep reinforcement learning based recommendation with explicit user-item interactions modeling," *arXiv preprint arXiv:1810.12027*, 2018.
[83] Y. Liu, Y. Zhang, Q. Wu, C. Miao, L. Cui, B. Zhao, Y. Zhao, and L. Guan, "Diversity-promoting deep reinforcement learning for interactive recommendation," *arXiv preprint arXiv:1903.07826*, 2019.
[84] X. Chen, S. Li, H. Li, S. Jiang, Y. Qi, and L. Song, "Generative adversarial user model for reinforcement learning based recommendation system," in *ICML*, 2019, pp. 1052–1061.
[85] J. Shi, Y. Yu, Q. Da, S. Chen, and A. Zeng, "Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning," in *AAAI*, vol. 33, 2019, pp. 4902–4909.
[86] L. Zou, L. Xia, P. Du, Z. Zhang, T. Bai, W. Liu, J. Nie, and D. Yin, "Pseudo dyna-q: A reinforcement learning framework for interactive recommendation," in *WSDM*. ACM, 2020.
[87] F. Liu, H. Guo, X. Li, R. Tang, Y. Ye, and X. He, "End-to-end deep reinforcement learning based recommendation with supervised embedding," in *WSDM*, 2020, pp. 384–392.



Yu Lei is currently a Ph.D. candidate in the Department of Computing, The Hong Kong Polytechnic University. He received his M.S. degree in Computer Science from the School of Computer Science and Technology, Jilin University, China, in 2014. His main research interests include recommender systems, reinforcement learning, collaborative filtering/ranking and social network analysis.



Zhitao Wang is currently a PhD candidate in the Department of Computing, The Hong Kong Polytechnic University. He received his BEng degree and MSc degree in computer science from Northwestern Polytechnical University, China, in 2012 and 2015, respectively. His main research interests include online social network analysis, information diffusion and network embedding.

Wenjie Li is currently an associate professor in the Department of Computing, The Hong Kong Polytechnic University. She received her PhD degree in systems engineering and engineering management from the Chinese University of Hong Kong, Hong Kong, in 1997. Her main research interests include online social network analysis, natural language processing and document summarization.

Hongbin Pei is currently a Ph.D. candidate in the Shool of Computer Science and Technology, Jilin University. He received the M.S. and B.S. degrees from the Shool of Computer Science and Technology, Jilin University, in 2015 and 2012, respectively. His research interests include spatiotemporal data mining and network data analysis with applications for health informatics and intelligent transportation.

Quanyu Dai is currently a Ph.D. candidate in the Department of Computing, The Hong Kong Polytechnic University. He received the B.Eng. degree in the Department of Electronic Engineering, Shanghai Jiao Tong University. His research interests include representation learning, transfer learning, deep learning, and adversarial learning for graph-structured data. He has publications appeared in the top-tier conferences such as AAAI and WWW.