Reinforcement Learning based Recommendation with Graph Convolutional Q-network

Yu Lei[1], Hongbin Pei[2], Hanqi Yan[3], Wenjie Li[1]
[1]The Hong Kong Polytechnic University, Hong Kong, China
[2]Jilin University, Changchun, China
[3]Peking University, Beijing, China
{csylei,cswjli}@comp.polyu.edu.hk;peihb15@mails.jlu.edu.cn;hanqi.yan@pku.edu.cn

ABSTRACT

Reinforcement learning (RL) has been successfully applied to recommender systems. However, the existing RL-based recommendation methods are limited by their unstructured state/action representations. To address this limitation, we propose a novel way that builds high-quality graph-structured states/actions according to the user-item bipartite graph. More specifically, we develop an end-to-end RL agent, termed Graph Convolutional Q-network (GCQN), which is able to learn effective recommendation policies based on the inputs of the proposed graph-structured representations. We show that GCQN achieves significant performance margins over the existing methods, across different datasets and task settings.

CCS CONCEPTS

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

KEYWORDS

Reinforcement learning based recommendation; Graph convolutional Q-network; Reinforcement learning; Recommender Systems

ACM Reference Format:
Yu Lei, Hongbin Pei, Hanqi Yan, Wenjie Li. 2020. Reinforcement Learning based Recommendation with Graph Convolutional Q-network. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China.* ACM, NY, NY, USA, 4 pages. https://doi.org/10.1145/3397271.3401237

1 INTRODUCTION

Reinforcement learning (RL) [13] is a promising approach to recommendation, which aims to build an agent that adaptively recommends potentially interesting items to users in a sequential manner. Recently, researchers proposed deep RL based recommendation agents, which use deep neural networks to approximate the optimal action-value functions or policies, and show great potential in a variety of recommendation domains ranging from news feeds to E-commerce sites [2, 16, 17]. However, there is an important problem that has been rarely noticed and investigated in these prior works. That is, *how to effectively represent the states and actions for an RL recommendation agent, according to specific characteristics in recommender systems, e.g., user-item bipartite graph?* In some works [17], the states (actions) are represented by some handcrafted features of users (items). Other researchers [16] leverage neural networks to learn high-level vector representations from the raw states and actions. A common weakness of these existing RL-based methods is that *they represent the states/actions of each individual user in isolation, such that the underlying relationships (e.g., behavioral similarities) between different users are not effectively explored and exploited.* This weakness significantly limits the performance of existing methods in learning effective recommendation policies for the entire user community.

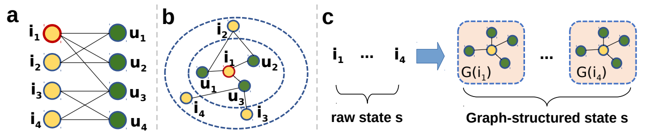


Figure 1: A toy example to illustrate how to build a graph-structured state for target user u. (a) The whole user-item bipartite graph G. (b) The sub-graph $G(i_1)$ that consists of item i_1 and its neighborhood in G. (c) Building a graph-structured state $s = \{G(i_1), ..., G(i_4)\}$ based on the raw state $s = \{i_1, ..., i_4\}$.

In this work, we overcome the weakness by designing graph-structured representations of states and actions. We start by following the existing work [14] to define the raw state as an item sequence observed currently. We then build a graph-structured state based on the raw item sequence by replacing each item i in the sequence with a specific sub-graph $G(i)$ that consists of item i and its neighborhood in the whole user-item bipartite graph G. Figure 1 illustrates how to build a graph-structured state for target user u through a toy example. *Such graph-structured state has two notable qualities: (1) it captures the rich structural information in user-item bipartite graph, which enables the agent to discover collaborative preferences from target user u's neighbors (e.g., u_1, u_2, u_3); and (2) it preserves the sequential information in original item sequence, which enables the agent to model the dynamic preferences of target user u.* In the same way, each action can be represented by a specific sub-graph instead of a single item.

We develop an end-to-end agent, termed Graph Convolutional Q-network (GCQN), which is able to transform the graph-structured states and actions to high-level vector representations and predict the action-values based on them (see Figure 2). More specifically, we propose a variant of graph convolutional network (GCN) to extract information from the graph-structured states and actions, and output some graph-aware vector representations. We then take advantage of the gated recurrent unit (GRU) [3] with a self-attention mechanism [10], to process the sequence of graph-aware representations and produce a higher-level vector representation of the state. Finally, the concatenation of the high-level state representation and action representation is fed into a multilayer perceptron (MLP) to predict the action-value of the state-action pair. We empirically validate GCQN by conducting extensive experiments on three real-world datasets. The results show that GCQN is effective and robust, which achieves significant performance gains over state-of-the-art baselines across different datasets and task settings.

2 PRELIMINARIES

We formulate the RL-based recommendation problem as follows. Suppose we have a recommender system with a set of m users $\mathcal{U} = \{1, ..., m\}$, a set of n items $\mathcal{I} = \{1, ..., n\}$, and an observed user-item implicit feedback matrix $Y \in \mathbb{R}^{m \times n}$, where $y_{ui} = 1$ if user u gives a positive feedback on item i (clicking, watching, etc.), and $y_{ui} = 0$ otherwise. We consider an episodic RL task [13], where in each episode, a recommendation agent interacts with a target user u at discrete time steps $t = 0, ..., T-1$. At each time step t, the agent observes a state s_t (the current situation the agent faces), and accordingly takes an action a_t (i.e., recommends an item $a_t \in \mathcal{I}$) based on its policy π (indicating how to choose actions given states). One step later, as a consequence of its action a_t, the agent receives a reward $r_{t+1} = y_{ua_t}$ from user u and observes next state s_{t+1}.

Given the data of training users $\mathcal{U}_{train} \subset \mathcal{U}$, our goal is to learn a policy that maximizes the cumulative rewards received in a T-step episode, $\sum_{t=0}^{T-1} r_{t+1}$, for testing users $\mathcal{U}_{test} = \mathcal{U} \setminus \mathcal{U}_{train}$. To estimate the policy, in this work we follow a well-adopted approach that uses a neural network $Q(s, a; \theta)$ (i.e., Q-network) to approximate the optimal action-value function $Q^*(s, a)$ (corresponding to an optimal policy π^*) [12, 13].

3 GRAPH CONVOLUTIONAL Q-NETWORK

As mentioned previously, we define the raw state as $\{a_0, ..., a_{t-1}\}$, a sequence of items that target user u has consumed before time step t, and define the raw action as a unique item a. We then define the graph-structured state as $\{G(a_0), ..., G(a_{t-1})\}$ and the graph-structured action as $G(a)$, where each sub-graph $G(i)$ consists of item i and its neighborhood in the whole user-item bipartite graph G[1] (see Figure 1). As shown in Figure 2, the proposed GCQN takes the graph-structured representations of state-action pair (s_t, a) as input, and outputs the predicted action-value $Q(s_t, a)$. We will elaborate on GCQN in the following.

3.1 Model Architecture

Embedding Layer. We first map all users and items to a low-dimensional vector space. Each user u and each item i is described by a unique embedding vector $\mathbf{e}_u \in \mathbb{R}^d$ and $\mathbf{e}_i \in \mathbb{R}^d$, respectively. The embeddings of users and items are implemented as two simple lookup tables: $\mathbf{E}_u = [\mathbf{e}_1, ..., \mathbf{e}_m]$ and $\mathbf{E}_i = [\mathbf{e}_1, ..., \mathbf{e}_n]$. These embeddings are treated as the raw features of nodes in the graphs, which are randomly initialized and trained in an end-to-end fashion.

GCN Layer. We develop an variant of GCN by taking advantage of the ideas of both GraphSage [4] and GAT [15]. The goal of this

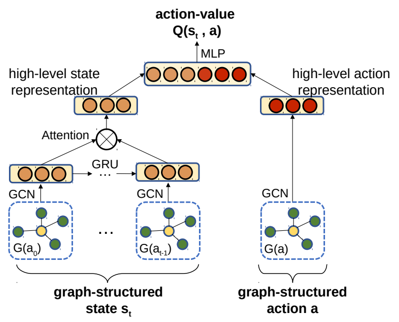

action-value
Q(s_t , a)
MLP
high-level state representation
high-level action representation
Attention
GRU
GCN
GCN
GCN
G(a_0)
G(a_{t-1})
G(a)
graph-structured state s_t
graph-structured action a

Figure 2: Overview of the GCQN model.

GCN module is to process each sub-graph $G(i)$, and produce a graph-aware representation of item i, $\mathbf{x}_i \in \mathbb{R}^d$:

$$\mathbf{x}_i \leftarrow \text{relu}(\mathbf{W}_{fc}[\mathbf{e}_i \oplus \mathbf{e}_{\mathcal{N}(i)}] + \mathbf{b}_{fc}), \tag{1}$$

where $\mathbf{W}_{fc} \in \mathbb{R}^{d \times 2d}$ and $\mathbf{b}_{fc} \in \mathbb{R}^d$ are the trainable weights and biases of a fully connected (FC) layer, $\mathbf{e}_i$ is the embedding of item i, $\oplus$ denotes concatenation, and $\mathbf{e}_{\mathcal{N}(i)} \in \mathbb{R}^d$ is the neighborhood vector computed by an attention-based aggregator AGG^{att}:

$$\mathbf{e}_{\mathcal{N}(i)} \leftarrow \sum\nolimits_{w \in \mathcal{N}(i)} \alpha_{iw} \mathbf{e}_w, \tag{2}$$

where $\mathcal{N}(i)$ denotes the set of the 1-hop neighbors of item i in $G(i)$, $\mathbf{e}_w$ is the embedding of user w, and α_{iw} is the attention score that determines how much feature information of user w will be passed to item i. We adopt the Concat attention mechanism [11] to compute the attention score α_{iw}:

$$\alpha_{iw} \leftarrow \frac{\exp\left(\mathbf{w}_a^\top \tanh(\mathbf{W}_a[\mathbf{e}_i \oplus \mathbf{e}_w])\right)}{\sum_{v \in \mathcal{N}(i)} \exp\left(\mathbf{w}_a^\top \tanh(\mathbf{W}_a[\mathbf{e}_i \oplus \mathbf{e}_v])\right)}, \tag{3}$$

where $\mathbf{W}_a \in \mathbb{R}^{d \times 2d}$ and $\mathbf{w}_a \in \mathbb{R}^d$ are trainable weights for attention, and $\cdot^\top$ is the transpose operation.

In a real-word recommender system, the size of $\mathcal{N}(i)$ may vary dramatically over different items i (e.g., follow the long-tail distribution). To make the computation more efficient, as many existing works did [4], we uniformly sample a fixed-size set of user neighbors for each item i, instead of using its full neighborhood. That is, we redefine $\mathcal{N}(i)$ as a fixed L-size, uniform sample from the full set $\mathcal{N}^+(i) = \{u : y_{ui} = 1\}$. Note that $\mathcal{N}(i)$ is formed with different uniform samples at each iteration during training, and will contain duplicates when $L > |\mathcal{N}^+(i)|$. In our experiments, we also examined the mean-based aggregator AGG^{mean} and the pooling-based aggregator AGG^{pool} proposed in [4]. However, these two aggregators did not show comparable performance against our attention-based one (see Table 3 for comparison), as they failed to model the different influence strength from each user $v \in \mathcal{N}(i)$ to item i.

By applying the GCN module to the action graph $G(a)$ and the state graphs $\{G(a_0), ..., G(a_{t-1})\}$, we obtain a graph-aware vector representation $\mathbf{x}_a$ of action a, and a sequence of graph-aware vector representations $\mathbf{x}(s_t) = \{\mathbf{x}_{a_0}, ..., \mathbf{x}_{a_{t-1}}\}$ of state s_t, respectively.

GRU Layer. To model the sequential information in the state, as the exiting work did [16], we leverage a gated recurrent unit (GRU)[2] to further process the graph-aware state representation $\mathbf{x}(s_t) = \{\mathbf{x}_{a_0}, ..., \mathbf{x}_{a_{t-1}}\}$, which can be simply denoted by $\{\mathbf{x}_0, ..., \mathbf{x}_{t-1}\}$ for

[1] In practice, the graph G is constructed based only on training users' data.

[2] We chose GRU instead of long-short term memory (LSTM) and simple RNN because GRU has shown advantages over them in many recommendation tasks [7, 16].

notational convenience. The goal of this GRU module is to transform $\{\mathbf{x}_0, ..., \mathbf{x}_{t-1}\}$ to a sequence of hidden vectors $\{\mathbf{h}_0, ..., \mathbf{h}_{t-1}\}$, where $\mathbf{h}_j \in \mathbb{R}^d$ is abstractly computed as: $\mathbf{h}_j \leftarrow \text{GRU}(\mathbf{h}_{j-1}, \mathbf{x}_j)$. Furthermore, we employ a self-attention mechanism [10] to capture the importance of different items in the state, and summarize a final state representation $\mathbf{h}_{s_t} \leftarrow \sum_{j=0}^{t-1} \beta_j \mathbf{h}_j$, where β_j is the attention score that indicates how much feature information of item a_j will be extracted, which is computed by: $\beta_j \leftarrow \frac{\exp\left(\mathbf{w}_{sa}^\top \tanh(\mathbf{W}_{sa}\mathbf{h}_j)\right)}{\sum_{l=0}^{t-1} \exp\left(\mathbf{w}_{sa}^\top \tanh(\mathbf{W}_{sa}\mathbf{h}_l)\right)}$, where $\mathbf{W}_{sa} \in \mathbb{R}^{d\times d}$ and $\mathbf{w}_{sa} \in \mathbb{R}^d$ are trainable weights in the self-attention. With this attention mechanism, the GRU layer is able to autonomously select more important features from the input sequence, and thus help the agent capture the user's dynamic preference at each time step.

MLP Layers. Once the state vector representation $\mathbf{h}_{s_t} \in \mathbb{R}^d$ and the action vector representation $\mathbf{x}_a \in \mathbb{R}^d$ are ready, we employ a multilayer perceptron (MLP) (with architecture $2d \to \cdots \to 1$) to fuse useful feature information from both of them and predict the final action-value $Q(s_t, a)$.

3.2 Model Training

To train GCQN, we minimize the following loss function:

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

where θ denote all trainable parameters of the Q-network, $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$ is the target for current iteration, γ is the discount factor that balances the importance between future rewards and immediate rewards, and θ^- are the Q-network parameters from previous iteration, which are held fixed when performing optimization. In practice, instead of optimizing the full expectations in the loss function, a more convenient way is to perform stochastic gradient descent (SGD) on a sampled transition (s, a, r, s'):

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

The training algorithm of GCQN is presented in Algorithm 1. To make the Q-network converge well, in each episode, we uniformly sample a user u from $\mathcal{U}_{train}$ 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.

Algorithm 1: Training GCQN

```
Input: training user set $\mathcal{U}_{train}$, feedback data $Y$
Output: trained Q-network $Q$
1 for episode = 1, ..., $N$ do
2     Uniformly sample a target user $u$ from $\mathcal{U}_{train}$
3     Initialize $s_0 = \{i_c\}$, where $i_c$ is a random item
4     for $t = 0, ..., T-1$ do
5         Pick the $\epsilon$-greedy item $a_t$ w.r.t. $Q(s_t, a; \theta)$
6         Set $r_{t+1} = y_{ua_t}$ and $s_{t+1} = s_t \cup \{a_t\}$
7         Update $Q$'s weights $\theta$ according to Equation 5
8     end
9 end
```

4 EXPERIMENTS

4.1 Experimental Setup

Datasets. We use three public recommendation datasets: **LastFM** [1], **ML1M** [5], and **Pinterest** [6]. All datasets contain user-item feedback data. Since we focus on implicit-feedback recommendations, we follow the common practice to treat users' diverse behaviors on items as a unified implicit positive feedback. To ensure there is enough data for evaluating RL agents, we first remove the items with fewer than 5 feedbacks for all datasets, and then remove the users with fewer than 5, 20 and 30 feedbacks for LastFM, ML1M and Pinterest, respectively (in proportion to the number of items in each dataset). The statistics of obtained datasets is given in Table 1.

Table 1: The statistics of datasets.

Statistics	LastFM	ML1M	Pinterest
#users	1,874	6,040	13,397
#items	2,828	3,416	9,359
#observed feedbacks	71,411	999,611	494,523

Evaluation Protocols. Since not all unobserved items are truly negative, we randomly select 1000 unobserved (u, i) pairs of user u as the negative feedbacks ($y_{ui} = 0$). In each T-step episode of user-agent interactions, the agent is forced to pick items from the available item set that consists of the 1000 sampled negative items and the observed positive items. We conduct experiments for cold-start scenario, which implies that the agent has no feedback data of target user at time step $t = 0$, i.e., at the beginning of each T-step episode. We split each dataset by randomly choosing 80% users as the training set $\mathcal{U}_{train}$, and the remaining 20% users as the testing set $\mathcal{U}_{test}$. We conduct each experiment on 5 data splits obtained with different random seeds. The evaluation metric we used is the mean of the rewards received in a T-step episode, which is equivalent to the ratio of positive items in the recommended items during the episode. The final metric is obtained by taking its average over all testing users and 5 data splits.

Baselines. To comparatively evaluate GCQN, we choose **five RL-based baselines**, which can be applied to our task with reasonable adaptions or extensions: **(1) DEERS** [16] is a DQN-based method that utilizes GRU to learn state representations from positive- and negative-feedback item sequences; **(2) LSTM-Q** learns the state representations from item sequences by using LSTM [8] and learns the action representations from item ids by using embedding layer; **(3) GRU-Q** is similar to LSTM-Q with the only difference of utilizing GRU [3] to learn state representations; **(4) AttLSTM-Q** extends LSTM-Q with a self-based attention mechanism [10]; **(5) AttGRU-Q** extends GRU-Q with the same self-based attention mechanism. All of the above baselines are Q-network agents. The major difference between them and our GCQN is that they only use RNNs to learn state representations and use a simple embedding layer to learn action representations. To apply them to our task, as well as for fair comparisons, we train these agents using the same Q-learning algorithm of GCQN (i.e., Algorithm 1). This enables us to focus on validating the efficacy of the proposed graph-structured representations. We also compare **three non-RL baselines** for reference: **(1) SVD** [9] is a matrix factorization model that uses the inner product of latent feature vectors to predict user-item relevance scores; **(2) Popular** picks items with most positive feedbacks, which is a simple but strong baseline in many recommendation tasks; **(3) Random** picks items randomly, which can be seen as an indicator that reveals the difficulty of the task itself.

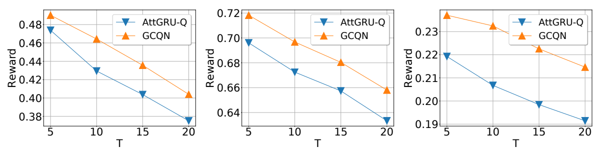
4.2 Results and Analysis

The comparison results of all methods, in terms of the average reward received in T-step episodes ($T = 20$), are reported in Table 2. The best performing method is highlighted in bold font. Our GCQN shows remarkable margins over the baselines on all datasets, including AttLSTM-Q and AttGRU-Q whose major differences between GCQN is that they do not use the graph-structured state/action representations. More specifically, the improvements of GCQN over the best performing baseline AttGRU-Q are about 7.7%, 3.9% and 12.5% on LastFM, ML1M and Pinterest datasets, respectively. *This clearly demonstrates that the proposed graph representations of states and actions are very helpful to learning RL-based recommendation policies, by effectively exploiting them with GCN networks.*

Table 2: Comparison for T-step recommendation ($T = 20$).

Type	Method	LastFM	ML1M	Pinterest
Non-RL	Random	0.036	0.123	0.034
	Popular	0.330	0.608	0.175
	SVD	0.151	0.285	0.084
RL	DEERS	0.243	0.511	0.095
	LSTM-Q	0.336	0.621	0.171
	GRU-Q	0.346	0.626	0.178
	AttLSTM-Q	0.354	0.632	0.181
	AttGRU-Q	0.375	0.633	0.191
Graph RL	GCQN	**0.404**	**0.658**	**0.215**

We also show the performance of GCQN when varying $T \in \{5, 10, 15, 20\}$, in comparison to the best performing baseline AttGRU-Q. The results are shown in Figure 3, from which we observe that GCQN consistently shows significant improvements over AttGRU-Q for different T. *This verifies again the effectiveness and robustness of GCQN, across different datasets and task settings.*



(a) LastFM **(b) ML1M** **(c) Pinterest**

Figure 3: Comparison for different $T \in \{5, 10, 15, 20\}$.

To study the impact of aggregator on GCQN, we design two variants of GCQN by replacing its attention-based aggregator $\mathrm{AGG}^{\mathrm{att}}$ with mean-based $\mathrm{AGG}^{\mathrm{mean}}$ and pooling-based $\mathrm{AGG}^{\mathrm{pool}}$ proposed in [4]. The comparison results in Table 3 show that the proposed $\mathrm{AGG}^{\mathrm{att}}$ outperforms the others by a significant margin. This verifies the efficacy of using attention mechanisms to capture different neighbors' features, instead of treating them equally.

Table 3: The results of GCQN with different aggregators.

AGG	LastFM		ML1M		Pinterest	
	$T = 10$	$T = 20$	$T = 10$	$T = 20$	$T = 10$	$T = 20$
mean	0.459	0.398	0.687	0.644	0.227	0.211
pool	0.458	0.400	0.691	0.648	0.221	0.211
att	**0.464**	**0.404**	**0.697**	**0.658**	**0.232**	**0.215**

We also investigate the impact of neighborhood sample size L on GCQN. The results of different $L \in \{2, 5, 10, 20\}$ are shown in Table 4. We observe that GCQN with $L = 10$ achieves the best performance (except for the case of $T = 20$ on Pinterest). This is because a too small L is not able to incorporate enough neighborhood information, while a too large L might bring noisy information.

Table 4: The results of GCQN with different sample size L.

L	LastFM		ML1M		Pinterest	
	$T = 10$	$T = 20$	$T = 10$	$T = 20$	$T = 10$	$T = 20$
2	0.458	0.401	0.694	0.652	0.226	0.209
5	0.461	0.403	0.695	0.652	0.228	**0.217**
10	**0.464**	**0.404**	**0.697**	**0.658**	**0.232**	0.215
20	0.458	0.401	0.693	0.651	0.230	0.212

5 CONCLUSIONS

We propose a novel idea that designs graph-structured states and actions for RL recommendation agents. To implement the idea, we develop an effective end-to-end RL agent, termed Graph Convolutional Q-network (GCQN), which is able to approximate the optimal action-value function based on the inputs of graph-structured representations, by successfully leveraging GCN and GRU to exploit the structural and sequential information, respectively. We have empirically validated the proposed idea, as well as the developed GCQN, by conducting solid experiments on real-world datasets.

ACKNOWLEDGMENTS

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

REFERENCES

[1] I. Cantador, P. L. Brusilovsky, and T. Kuflik. 2011. *HetRec2011*. ACM.
[2] 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.
[3] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).
[4] W. Hamilton, Z. Ying, and J. Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.
[5] F. M. Harper and J. A. Konstan. 2016. The movielens datasets: History and context. *TIIS* 5, 4 (2016), 19.
[6] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua. 2017. Neural collaborative filtering. In *WWW*. IW3C2, 173–182.
[7] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. 2016. Session-based recommendations with recurrent neural networks. In *ICLR*.
[8] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[9] Y. Koren, R. Bell, and C. Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.
[10] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio. 2017. A structured self-attentive sentence embedding. In *ICLR*.
[11] M. Luong, H. Pham, and C. D. Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).
[12] 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. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.
[13] R. S. Sutton and A. G. Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.
[14] N. Taghipour, A. Kardan, and S. S. Ghidary. 2007. Usage-based web recommendations: a reinforcement learning approach. In *RecSys*. ACM, 113–120.
[15] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. 2017. Graph attention networks. *arXiv* (2017).
[16] 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.
[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.