Trends Tracking Memory Recommender Networks for Product Recommendations in E-commerce

Zhi LI[†], Boqi GAO[†], Daichi AMAGATA[†], Takuya MAEKAWA[†], Takahiro HARA[†], Hao NIU[††], Kei YONEKAWA[††], and Mori KUROKAWA[††]

† Graduate School of Information Science and Technology, Osaka University
1-5, Yamadaoka, Suita, Osaka, 565-0871, Japan
†† Integrated Analysis Platform Laboratory, KDDI Research, Inc.
Chiyoda-ku, Tokyo, 102-8460, Japan
E-mail: †{li.zhi,gao.boqi,amagata.daichi,maekawa,hara}@ist.osaka-u.ac.jp,
††{ha-niu,ke-yonekawa,mo-kurokawa}@kddi-research.jp

Abstract In the online e-commerce recommendation field, changes of the market trends and user preferences become a new challenge to existing recommender systems. To tackle this challenge, prior studies have proposed neural memory recommender networks (NMRNs), which have external memories to capture the changes of the market trends and user preferences. Unfortunately, NMRNs cannot precisely capture the changes and suffer from computational inefficiency. This paper solves these problems and proposes a market-based memory updating strategy. This strategy enables the memory networks to flexibly adjust its memory updating frequency. Besides, we utilize a naive pairwise Hinge-loss to enable the memory networks to be trained in parallel by a mini-batch of user-item interactions. Experimental results show the superiority of our proposed methods in terms of recommendation accuracy and computational efficiency.

Key words Recommender Systems, Online E-commerce Recommendation, Memory Network, Market Trends, Dynamic User Preferences, Generative Adversarial Networks,

1. Introduction

With the explosive growth of available online information, users spend a long time to select their suitable items from many products, movies, and restaurants. A recommender system is an intuitive and effective choice to defend against this consumer over-choice [1]. Moreover, utilizing a recommender system becomes a general way to improve user experiences and supplier profits. Recently, such systems have been studied and applied in a variety of fields, *e.g.*, online movie recommendations in Netflix [2], video recommendations on YouTube [3] and academic collaborator recommendations [4].

In general, recommender systems can be categorized into: collaborative filtering recommender systems [5] and content based recommender systems [4]. Collaborative filtering ones generate recommendation lists based on user-item interaction records, while content-based ones are trained on the user and item side information (*e.g.*, description of items including texts, images, and videos). Since content-based recommender systems strongly rely on domain-specific side information, it is hard to design robust recommender systems. Therefore, to build a robust recommender system that does not rely on side information, many studies focus on collaborative filtering recommender systems.

Traditional collaborative filtering recommender systems [5] learn recommendation lists by encoding users and items into a latent space and represent users and items with user preference vectors and item attributes vectors, respectively. Many studies assume that user preferences and item attributes are static. However, the information in the real-world is always dynamic. For example, [6] has introduced three temporal and dynamic factors in a movie recommendation field: changes in movie perceptions, seasonal changes, and user interests. Static preference-based recommender systems cannot properly process these dynamic factors, resulting in bad recommendation performance. As in the case of the movie domain, static recommender systems also perform poorly in online e-commerce platforms, in particular for the case of online discount sales [7]. This is because static recommender systems cannot track fast-changing trends, and e-commerce websites change their products frequently to follow the trends.

To remove the drawback of static recommender systems, many studies capture temporal information by using recurrent neural networks (RNN) and long short-term memory (LSTM) based models [6, 7, 8]. However, most of these RNN and LSTM-based models can serve only for session-based recommender systems [9], which can process only time-series data. RNN and LSTM-based models have limitations in capturing users' stable interests and inherent attributes of items because they update all the memory cells at each step. Besides, session-based recommender systems usually ignore users with few interactions to improve recommendation performance, because they have a poor performance for such users. Therefore, the session-based models cannot process the real-world highly sparse data well.

To relieve these restrictions, in [9], Wang *et al.* has developed a novel recommender system named neural memory recommender networks (NMRNs) based on key-value memory networks (KV-MemNN) [10]. Different from RNN and LSTM, KV-MemNN can read and write a part of memories, which means both the users' long-term stable preferences and short-term dynamic interests can be captured. However, the memory of NMRN is frequently updated for each user-item interaction. This incurs two severe problems; i) the learning speed is slow due to a large amount of memory update operations, and ii) the accuracy of recommendation is not high enough because NMRN is too sensitive to the latest trends (NMRN tracks the trends by utilizing the most recent item). Therefore, it ignores valuable old trends.

In this paper, we overcome the above-mentioned problems of NMRN. We propose a market-based memory update strategy for NMRN. In our strategy, instead of directly applying the most recent item to update the memory, we create a market vector which is aggregated by a mini-batch [11] of items to represent the changes of the market trends, and we update the memory with this market vector. With our network structure, the NMRN can be trained in parallel by mini-batch. Therefore, the training speed can be extensively accelerated. Furthermore, updating the memory of NMRN per mini-batch can reduce the influence of a single user-item interaction. The sensitivity of the NMRN thus is decreased, and the network can utilize valuable old trends, resulting in a high recommendation accuracy.

Our main contributions of this paper are summarized as follows.

- We improve the network structure of NMRN and propose a market-based memory update strategy for NMRN. In particular, we create a market vector to represent recent market trends. This improvement enables NMRN to be trained in parallel by mini-batch of user-item interactions.
- We conduct extensive experiments based on our real-world e-commerce dataset. The experimental results show that our model significantly outperforms NMRN in terms of accuracy and computational efficiency.

The rest of this paper is organized as follows. We review related works in Section 2. Then, we introduce NMRN in Section 3. Next, our proposed framework is described in Section 4. 1 The details of the experiments are reported in Section 5. Finally, this paper is summarized in Section 6.

2. Related Work

2. 1 Collaborative filtering recommender systems

Basic collaborative filtering recommender systems generate recommendation lists based on user-item historical interactions, either explicit (*e.g.*, previous ratings) or implicit feedback (*e.g.*, browsing history). One of the most famous methods is matrix factorization (MF) [5, 12]. It learns latent vectors which can represent static user interests and item inherent attributes from user-item interaction records. However, temporal and dynamic information are significant factors when designing personalized recommender systems, in particular, real-world online e-commerce platforms. Therefore, [13, 14, 15] have proposed session-based models, which can catch the sequence of information from users' historical behavior sessions by utilizing RNNs or LSTMs. However, they can process only time-series data and they ignore users with few interactions.

2. 2 Memory networks

Neural memory networks (MemNNs) are novel learning models inspired by the recent advances of modern computer architectures [16]. MemNNs can flexibly manage their memories because they read and write a part of memory components [17]. In [18], Weston *et al.* firstly proposed the concept of MemNNs. They have proved that MemNNs outperform RNN because the representation ability of MemNNs is higher than that of RNN and they can capture a trend changes. Later on, MemNNs have been utilized in many applications such as question answering [10, 19] and knowledge tracking [20]. MemNNs have also been utilized in recommender systems. In [9], Wang *et al.* firstly utilized MemNNs to build recommender systems. Their MemNN-based model can track market trends and capture the inherent stable pattern of users. However, their model has low computational efficiency because it updates its memory based on a single item. Online e-commerce recommendations require models with high computational efficiency to improve their profits. Therefore, a model with high computational efficiency should be considered.

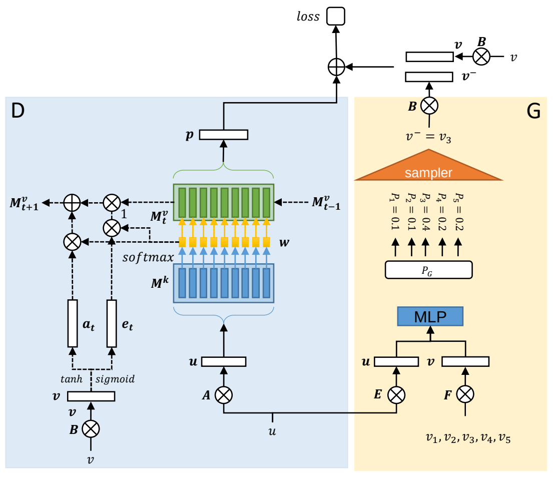

Figure 1 The architecture of NMRN. NMRN contains a discriminator, which measures the similarity between users and items, and a generator, which selects negative items that are informative for users.

3. Neural memory recommender network (NMRN)

In this section, we describe the architecture of the original NMRN, which is shown in Figure 1. NMRN is a recommender system based on KV-MemNN and generative adversarial networks (GAN) [21], which are composed of a discriminator (D) and a generator (G). The generator of NMRN is a Multi-Layer Perceptron (MLP), which samples informative negative items for a specific user. The discriminator of NMRN is a key-value memory network, which measures similarities between users and items. After the discriminator creates the similarities between a particular user and all items, NMRN selects the most similar items to the user as the recommendation lists for her. Note that we denote matrices and vectors with bold capital letters and bold small letters, respectively.

3. 1 Discriminator preliminaries

The total numbers of users and items are denoted as N and M. Let an observed user-item interaction pair be denoted by (u, v). u is the one-hot representation of a user and v is the one-hot representation of an item. Then, the original NMRN obtains the user vector $\mathbf{u}$ by multiplying u with user embedding matrix $\mathbf{A}$ and the item vector $\mathbf{v}$ by multiplying v with item embedding matrix $\mathbf{B}$, where $\mathbf{u}$ and $\mathbf{v}$ are real-valued vectors with r dimensions, size of $\mathbf{A}$ and $\mathbf{B}$ is $N \times r$ and $M \times r$.

3. 2 Architecture of the discriminator

NMRN assumes that users have two categories of interests: static and dynamic interests. Static interests show long-term interests (*e.g.*, a color that a particular user always prefers), while dynamic interests show short-term interests (*e.g.*, a recent popular electronic product). Therefore, in the discriminator, two matrices are utilized to represent these two categories of interests. One of them is a key memory matrix $\mathbf{M}^k$ that represents the static interests, and the other is a value memory matrix $\mathbf{M}_t^v$ that represents the dynamic interests. The size of $\mathbf{M}^k$ and $\mathbf{M}_t^v$ is $L \times r$

We first explain $\mathbf{M}^k$. $\mathbf{M}^k$ represents existing latent factors about users' static interests, and different users have different tastes on these latent factors. The similarities between users and these existing latent factors are measured by the Euclidean distance:

$$sim_i = \left\|\mathbf{u} - \mathbf{M^k}(i)\right\| \tag{1}$$

where sim_i is the similarity between $\mathbf{u}$ and the i^{th} existing latent factor $\mathbf{M^k}(i)$. Then, an attention weight vector $\mathbf{w}$ with size L is obtained as follows (take the i^{th} one, $w(i)$, as an example):

$$w(i) = \text{Softmax}\left(-\text{sim}_i\right) \tag{2}$$

By applying negative similarities, the most similar memory slots generate the largest weight.

Then, we introduce $\mathbf{M}_t^v$, which stores dynamic latent factors (*e.g.*, trends and discounts). Different from a key memory matrix, the value memory matrix is updated with recent items over time. Then, a vector $\mathbf{p}$, which represents user preferences named *proxy preference vector*, is calculated by multiplying each value memory slot with corresponding attention weight:

$$\mathbf{p} = \sum_{i=1}^{L} w(i)\mathbf{M}_t^v(i) \tag{3}$$

After obtaining $\mathbf{p}$, the discriminator applies the Euclidean distance d to measure the similarities between a user u and an item v:

$$d(u, v) = ||\mathbf{p} - \mathbf{v}|| = \sqrt{\sum_i (p_i - v_i)^2}. \tag{4}$$

A user's proxy preference vector $\mathbf{p}$ should be similar to interacted items and different from non-interacted items. Pairwise Hinge-loss [22] can pull positive items (item vector) closer and push negative items (negative item vector) further from a specific user (proxy preference vector). NMRN applies a novel Weighted Approximate-Rank Pairwise (WARP) loss function [22] based on Hinge-loss. The WARP loss function is defined as follows:

$$\mathcal{L} = \sum_{(u,v)\in S} \sum_{v^- \sim \mathcal{V}_u^-} w_{u,v} * \left|m + d(u, v) - d\left(u, v^-\right)\right|_+ \tag{5}$$

where $|z|_+ = max(z, 0)$ denotes the standard Hinge-loss, m is a safety margin size which should be larger than zero, S is the user-item interaction data utilized for training, v^- is a negative item sampled by generator, and $\mathcal{V}_u^-$ is a subset of items that u has never interacted with. Besides, $w_{u,v}$ denotes the penalty of a positive item v:

$$w_{u,v} = \log\left(rank_{u,v} + 1\right) \tag{6}$$

where $rank_{u,v}$ denotes the rank of item v in u's recommendation list. Because ranking all items results in a high computational cost, NMRN ranks sampled negative items. The approximate rank is calculated as:

$$rank_{u,v} \approx \left\lfloor \frac{N-1}{N_{u,v}} \right\rfloor \tag{7}$$

Note that N is the total number of items and $N_{u,v}$ is the number of negative items that need to be drawn until v^- satisfies $d(u,v) - d\left(u, v^-\right) + m > 0$.

3. 3 Memory update of discriminator

As mentioned above, the value memory matrix $\mathbf{M}_t^v$ represents the dynamic interests of users at time t. Since markets and trends are always changing, the discriminator updates the value memory matrix to mimic the trends of the market and adapt to the changes in user preferences. Let $\mathbf{M}_{t+1}^v$ denote the value memory matrix after being updated. NMRN utilizes two steps, erasing and adding steps, to update the value memory matrix from $\mathbf{M}_t^v$ to $\mathbf{M}_{t+1}^v$.

In the erasing step, the value memory matrix $\mathbf{M}_t^v$ is partially erased and modified by an *erased vector* $\mathbf{e}_t$:

$$\tilde{\mathbf{M}}_{\mathrm{t+1}}^{\mathrm{v}}(i) = \mathbf{M}_{\mathrm{t}}^{\mathrm{v}}(i) \circ \left[\mathbf{I} - w(i)\mathbf{e}_{\mathbf{t}}\right] \tag{8}$$

$$\mathbf{e}_{\mathrm{t}} = \mathrm{Sigmoid}\left(\mathbf{W}_{\mathrm{e}}\mathbf{v} + \mathbf{b}_{\mathrm{e}}\right) \tag{9}$$

where $\mathbf{I}$ is a vector with each element being 1, $\circ$ is element-wise multiplication, $\mathbf{W}_e$ is a linear transformation matrix and $\mathbf{b}_e$ is a bias vector. In the adding step, an *add vector* $\mathbf{a}_t$ is calculated in a similar way:

$$\mathbf{a}_{\mathrm{t}} = \mathrm{Tanh}\left(\mathbf{W}_{\mathbf{a}}\mathbf{v} + \mathbf{b}_{\mathbf{a}}\right) \tag{10}$$

where $\mathbf{W}_a$ and $\mathbf{b}_a$ are linear transformation matrix and bias vector, respectively. Finally, the value memory matrix at time $t+1$ is obtained as follows:

$$\mathbf{M}_{\mathbf{t+1}}^{\mathbf{v}}(i) = \tilde{\mathbf{M}}_{\mathbf{t+1}}^{\mathbf{v}}(i) + w(i)\mathbf{a}_{\mathbf{t}} \tag{11}$$

In the training phase, NMRN randomly samples a mini-batch of interactions S_{batch} from S and updates $\mathbf{M}_t^v$ per single interaction. Different from updating memory, NMRN accumulates the gradient of a single interaction between S_{batch}. Then, NMRN updates its parameters by accumulated gradients.

3. 4 Generator

The goal of the generator is to generate plausible negative items that confuse the discriminator so that the discriminator can generate better-ranked recommendation lists. Therefore, the objective function is to maximize the expectation of the similarity measured by the discriminator. The loss function of the generator $\mathcal{L}_G$ is described as follows:

$$\mathcal{L}_G = \sum_{\substack{(u,v)\in\mathcal{S} \\ v^- \sim P_G\left(v^-|u,v\right)}} \mathbb{E}\left[-d_D\left(u, v^-\right)\right] \tag{12}$$

where $-d_D(u,v)$ denotes the similarity between a user u and an item v measured by the discriminator. The distribution $P_G\left(v^-|u,v\right)$ is calculated as:

$$P_G\left(v^-|u,v\right) = \frac{\exp\left(-d_G\left(u,v^-\right)\right)}{\sum_{\bar{v}\in\mathcal{V}_u^-}\exp\left(-d_G(u,\bar{v})\right)} \tag{13}$$

where $d_G(u,v)$ is the Euclidean distance between user u and item v calculated by the generator. The transformation matrices $\mathbf{E}$ and $\mathbf{F}$ encode users and items to embedded vectors. Embedded user and item vectors share the same MLP to ensure that they are projected to the same space. $\mathcal{V}_u^-$ is a subset of all items that have no with user u, and $\bar{v}$, which denotes a candidate of negative items, is selected by uniformly at random to reduce computation time. The generator utilizes a policy gradient based reinforcement learning to optimize $\mathcal{L}_G$. Therefore, the gradient of $\mathcal{L}_G$ is calculated as:

$$\begin{aligned} \nabla_{\theta_G}\mathcal{L}_G &= \sum_{(u,v)\in\mathcal{S}} \mathbb{E}_{v^- \sim P_G}\left[-d_D\left(u,v^-\right)\right. \\ &\quad \left.\nabla_{\theta_G}\log P_G\left(v^-|u,v\right)\right] \\ &\simeq \sum_{(u,v)\in\mathcal{S}} \frac{1}{T} \sum_{v_i^- \sim P_G, i=1}^{T} \left[-d_D\left(u_i, v_i^-\right)\right. \\ &\quad \left.\nabla_{\theta_G}\log P_G\left(v_i^-|u,v\right)\right] \end{aligned} \tag{14}$$

3. 5 Top-*k* recommendations

To generate recommendation lists for users, all the similarities between users and items are calculated by the discriminator. Then, to generate a recommendation list for a specific user, items, which have no interaction with this user, are ranked by calculated similarity. Finally, the k most similar items are recommended to this user.

4. Market based neural memory recommender network (MB-NMRN)

In this section, we present the details of our market-based memory update strategy, which enables NMRN to be more flexible to adjust the update frequency of its memory. By combining the mini-batch memory update strategy and a naive pairwise Hinge-loss, MB-NMRN can compute a mini-batch of user-item interactions in parallel.

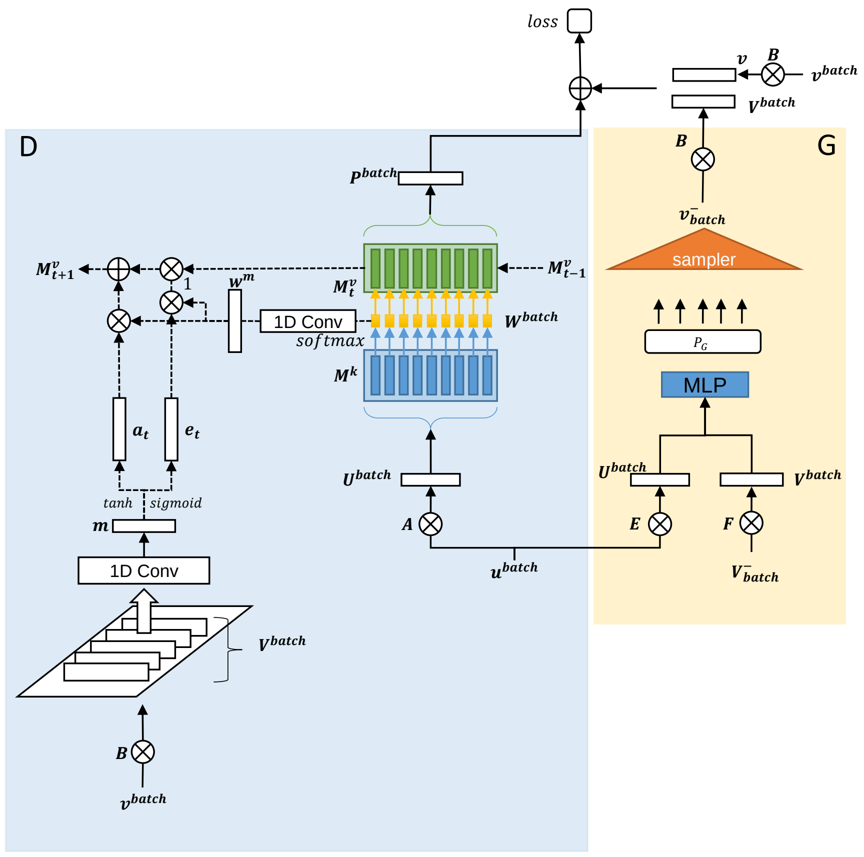

Figure 2 The architecture of MB-NMRN, which computes mini-batch of user-item interactions in parallel

4.1 Market based memory update strategy

Instead of updating value memory per user-item interaction by the most recent item, we utilize a one-dimensional convolutional layer (conv1D) to aggregate item vectors from a mini-batch to form a *market vector*, which represents the recent trends of the market. Then we update the value memory with this market vector per mini-batch. Our method can accelerate the training efficiency and balance the static and dynamic interests.

Figure 2 illustrates the structure of MB-NMRN. In this figure, $\mathbf{u}^{batch}$ denotes a batch of users (*i.e.*, a mini-batch of users) and $\mathbf{v}^{batch}$ denotes a batch of items. Then, MB-NMRN obtains the matrix of this batch of items $\mathbf{V}^{batch}$ by multiplying $\mathbf{v}^{batch}$ with item embedding matrix $\mathbf{B}$ and the matrix of this batch of items $\mathbf{U}^{batch}$ by multiplying $\mathbf{u}^{batch}$ with user embedding matrix $\mathbf{A}$. $\mathbf{V}^{-}_{batch}$ denotes a batch of candidate negative item lists and $\mathbf{m}$ denotes a market vector, which is calculated as follows:

$$\mathbf{m} = \mathbf{b_c} + \text{conv1D}\left(\mathbf{w}_c, \mathbf{V}^{batch}\right) \tag{15}$$

where $\mathbf{b}_c$ and $\mathbf{w}_c$ are defined as the bias and weight vectors of conv1D, respectively. Besides, an attention of the market, denoted by $\mathbf{w}^m$, is calculated from an attention matrix $\mathbf{W}^{batch}$, which is composed of a mini-batch of item attentions. The equation of $\mathbf{w}^m$ is described below:

$$\mathbf{w}^m = \mathbf{b}_c + \text{conv1D}\left(\mathbf{w}_c, \mathbf{W}^{batch}\right) \tag{16}$$

In our case, a market vector $\mathbf{m}$ and a market attention $\mathbf{w}^m$ are calculated by the same conv1D layer. This is because utilizing the same conv1D ensures that a specific item vector and its attention vector are matched (*i.e.*, an item, which has a large weight, makes its attention also has a large weight). Then, the value memory matrix $\mathbf{M}^v_t$ is updated by the erasing and adding steps described above.

Similar to Equations 8, 9, 10, and 11, the following equations show the process of updating $\mathbf{M}^v_t$ by our proposed strategy:

$$\begin{aligned}
\tilde{\mathbf{M}}^{\mathrm{v}}_{\mathrm{t+1}}(i) &= \mathbf{M}^{\mathrm{v}}_{\mathrm{t}}(i) \circ [\mathbf{I} - w^m(i)\mathbf{e_t}] \\
\mathbf{e}_{\mathrm{t}} &= \text{Sigmoid}\left(\mathbf{W}_{\mathrm{e}}\mathbf{m} + \mathbf{b}_{\mathrm{e}}\right) \\
\mathbf{a}_{\mathrm{t}} &= \text{Tanh}\left(\mathbf{W_a m} + \mathbf{b_a}\right) \\
\mathbf{M^v_{t+1}}(i) &= \tilde{\mathbf{M}}\mathbf{^v_{t+1}}(i) + w^m(i)\mathbf{a_t}
\end{aligned} \tag{17}$$

Our proposed strategy enables the updating frequency of the value memory to be adjustable by changing the mini-batch size.

4.2 A naive pairwise Hinge-loss

Deep neural networks (DNN) generally utilize mini-batches (*i.e.*, a set of data points) to train themselves in parallel. However, existing MemNNs [9,10,18] lose this parallel training structure because of locally updating their memory per data point. Therefore, they are computationally inefficient.

To address this drawback, we combine the mini-batch memory update strategy and a naive pairwise Hinge-loss $\mathcal{L}$. The mini-batch memory update strategy makes all the users within one mini-batch use the same memory to calculate their proxy preference vectors. And the naive pairwise Hinge-loss only samples one negative item for a user, which makes the parallel framework of discriminator simpler. Due to this, our modified memory network based discriminator can keep matrix parallel computation as well as other DNN architectures. The naive pairwise Hinge-loss $\mathcal{L}$ is calculated as follows:

$$\mathcal{L} = \sum_{\substack{(\mathbf{u}^{batch},\mathbf{v}^{batch})\in S \\ \mathbf{v}^{-}_{batch}\sim\mathcal{V}^{-}_{u}}} [m + \mathbf{d}(\mathbf{u}^{batch}, \mathbf{v}^{batch}) - \mathbf{d}\left(\mathbf{u}^{batch}, \mathbf{v}^{-}_{batch}\right)]_{+} \tag{18}$$

The difference between Equations 18 and 5 is that in Equation 18, we do not calculate $w_{u,v}$ and leave the negative item sampling procedure to the generator, because sampling more than one negative item for one user complicates the design of parallel model and have no contribution to the performance empirically. Compared with existing MemNNs, our proposed strategy has higher computational efficiency. This is because we update the memory by mini-batches and train a set of data points in parallel.

5. Experiment

In this section, we describe the details of our experiments. We first introduce the setting of our experiments. Then, we present the recommendation performances and computational efficiencies of the evaluation methods.

5.1 Experimental Settings

Dataset. We adopt a dataset from a real-world online e-commerce platform that provides items and services (*e.g.* home electronics, make-ups, and travel services). This dataset includes users' purchase records from *2017-05-11* to *2017-06-11*. We divide purchase records into three parts: *2017-05-11* to *2017-6-09* as training set, *2017-06-10* as validation set, *2017-06-11* as testing set. For new users and items which have no interaction information in the training set, we delete these new users and items from validation and testing sets. The details of the three data sets are summarized in Table 1.

Table 1 Statistics of the dataset set

Dataset	#interactions	#users	#items	Sparsity(%)
Training set	88,267	7,121	163,959	99.974
Validation set	1,382	553	1,773	99.768
Test set	1,551	499	1,944	99.749

Table 2 Hyper-parameter setting

Hyper parameters	value
Dimension of user and item vector	32
Dimension of memory slot (both key and value)	64
Maximum number of negative items sampled for one user	100
Safety margin size	5
Mini-batch size	2048

Evaluation Criteria The evaluation metrics that we adopt are Hits@k and Recall@k.

Hits@k is widely applied in recommender systems fields [9, 23, 24, 25]. After generating recommendation lists of users by the method that we mentioned in Section 3, each user-item interaction record (u, v) in the test set (D^{test}) is checked. For a specific user u, if she has interacted with an item v during the testing phase and v in his recommendation lists, we obtain a *hit*, otherwise, we obtain a *miss*. The Hits@k is calculated as follows:

$$\text{Hits@}k = \frac{\#\text{hit@}k}{|\mathcal{D}^{\text{test}}|} \tag{19}$$

where $|D^{test}|$ is the number of interactions in the test set and $\#hit@k$ is the number of hits within test set.

Recall@k is also widely adopted in recommender systems area [26,27,28,29]. The Recall@k for one user is defined as:

$$\text{Recall }@k = \frac{\#\text{ of true items in the top-k list}}{\text{the total }\#\text{ of the true items}} \tag{20}$$

where the true items mean the items that a specific user has interaction record during test phase. We average Recall@k among users in the test set.

Evaluation methods. To investigate the effectiveness of our proposed method, we prepared the following methods.

- **NMRN [9].** This is a state-of-art recommender system, which captures both users' long-term stable preferences and short-term dynamic interests. Moreover, NMRN is the first method that utilizes generative model-based sampler to generate informative negative items.
- **MB-NMRN.** This is our proposed method.

Hyper-parameter Settings. All evaluation methods were implemented based on PyTorch(注1). MB-NMRN achieves its best performance with the hyper-parameters show in Table 2. The optimizer is Adam [30], and the learning rate is 0.001.

(注1) : https://pytorch.org

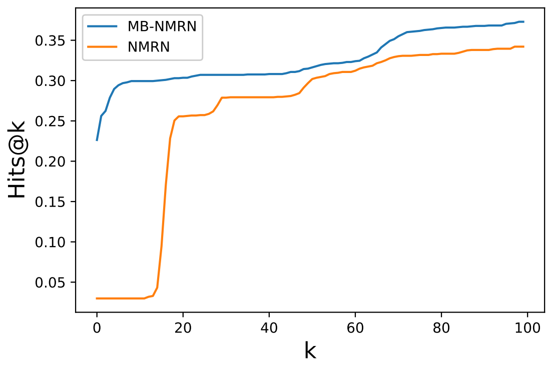

Figure 3　Hits@k of NMRN and MB-NMRN

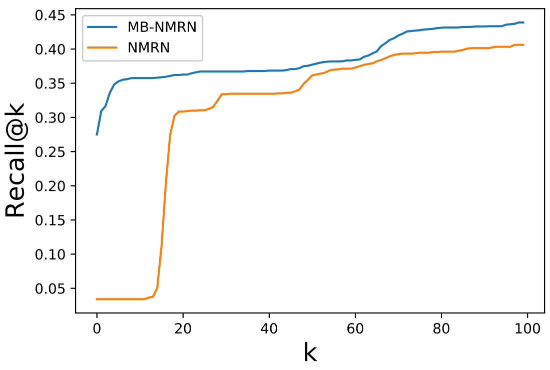

Figure 4　Recall@k of NMRN and MB-NMRN

We utilized a conv1D layer that has only 1 neural to compare the computational efficiency of MB-NMRN and NMRN. Other parameters are randomly initialized from a Gaussian distribution $\mathcal{N}(0, 0.01^2)$. For determining the best hyper-parameters, we tuned hyper-parameters based on the validation set. Instead of randomly sampling mini-batch user-item interactions from training set, which is applied in [9], we generated mini-batch in order of time strictly, to mimic the pattern of real-world online e-commerce platform.

5.2 Experimental Results

In this section, we report the recommendation performances and computational efficiencies of the evaluation methods.

Recommendation performance. Figures 3 and 4 show the results of Hits@k and Recall@k of the evaluation methods with k from 1 to 100, respectively. As Figures 3 and 4 show, MB-NMRN outperforms NMRN, in particular when k is smaller than 20. These results illustrate that compared with NMRN, MB-NMRN can better capture the changes of the market. Compared with directly using the most recent item to represent the changes of trends, the market vectors generated by MB-NMRN is more adaptive.

Computational efficiency. In this experiment, we test mini-batch size of [256, 512, 1024, 2048]. To avoid the influence of generator, we sample negative items uniformly at random for discriminator. Figure 5 shows the computational efficiency of NMRN and MB-NMRN. The vertical axis of this figure is running time until the discriminator loss of validation set is convergent, and the horizontal axis is mini-batch size. As Figure 5 shows, MB-NMRN outperforms NMRN in terms of computational efficiency.

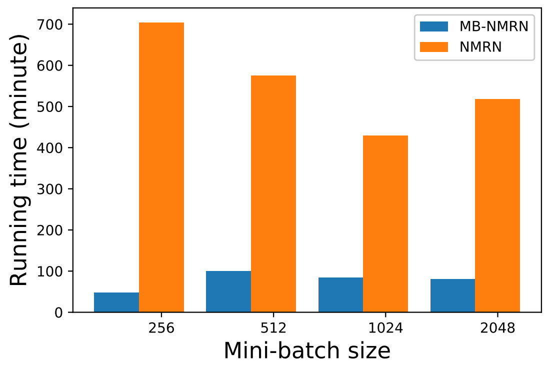

Figure 5　Running time of NMRN and MB-NMRN

6. Conclusion

In this study, we designed a market-based memory update strategy for NMRN. This strategy applies a conv1D layer to aggregate a mini-batch of items to form a market vector, and adopt this market vector to update the value memory of NMRN per mini-batch. Our strategy enables NMRN to adjust its memory update frequency. Besides, MB-NMRN can be trained by a mini-batch of user-item interactions in parallel. Our experimental results show that MB-NMRN outperforms NMRN in terms of recommendation performance and computational efficiency.

Acknowledgment

This research is partially supported by JST CREST Grant Number J181401085.

References

[1] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM CSUR*, 52(1):5, 2019.

[2] Carlos A Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM TMIS*, 6(4):13, 2016.

[3] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *RecSys*, pages 293–296, 2010.

[4] Zheng Liu, Xing Xie, and Lei Chen. Context-aware academic collaborator recommendation. In *KDD*, pages 1870–1879, 2018.

[5] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[6] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. Recurrent recommender networks. In *WSDM*, pages 495–503, 2017.

[7] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[8] Zhi Li, Hongke Zhao, Qi Liu, Zhenya Huang, Tao Mei, and Enhong Chen. Learning from history and present: Next-item recommendation via discriminatively exploiting user behaviors. In *KDD*, pages 1734–1743, 2018.

[9] Qinyong Wang, Hongzhi Yin, Zhiting Hu, Defu Lian, Hao Wang, and Zi Huang. Neural memory streaming recommender networks with adversarial training. In *KDD*, pages 2467–2475, 2018.

[10] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*, 2016.

[11] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

[12] Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In *NIPS*, pages 1257–1264, 2008.

[13] Dietmar Jannach and Malte Ludewig. When recurrent neural networks meet the neighborhood for session-based recommendation. In *RecSys*, pages 306–310, 2017.

[14] How Jing and Alexander J Smola. Neural survival recommender. In *WSDM*, pages 515–524, 2017.

[15] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. Stamp: short-term attention/memory priority model for session-based recommendation. In *KDD*, pages 1831–1839, 2018.

[16] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[17] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *ICML*, pages 1378–1387, 2016.

[18] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[19] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *NIPS*, pages 2440–2448, 2015.

[20] Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. *arXiv preprint arXiv:1612.03969*, 2016.

[21] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[22] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. Collaborative metric learning. In *WWW*, pages 193–201, 2017.

[23] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Recsys*, pages 39–46, 2010.

[24] Bo Hu and Martin Ester. Spatial topic modeling in online social media for location recommendation. In *Recsys*, pages 25–32, 2013.

[25] Weiqing Wang, Hongzhi Yin, Ling Chen, Yizhou Sun, Shazia Sadiq, and Xiaofang Zhou. Geo-sage: A geographical sparse additive generative model for spatial item recommendation. In *KDD*, pages 1255–1264, 2015.

[26] Ting Chen and Yizhou Sun. Task-guided and path-augmented heterogeneous network embedding for author identification. In *WSDM*, pages 295–304, 2017.

[27] Chong Wang and David M Blei. Collaborative topic modeling for recommending scientific articles. In *KDD*, pages 448–456, 2011.

[28] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *KDD*, pages 1235–1244, 2015.

[29] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *KDD*, pages 353–362, 2016.

[30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.