Neural Memory Streaming Recommender Networks with Adversarial Training

Qinyong Wang
School of Information Technology and Electrical Engineering, The University of Queensland
qinyong.wang@uq.edu.au

Hongzhi Yin*
School of Information Technology and Electrical Engineering, The University of Queensland
h.yin1@uq.edu.au

Zhiting Hu
Language Technologies Institute, Carnegie Mellon University
zhitingh@cs.cmu.edu

Defu Lian
School of Computer Science and Engineering, University of Electronic Science and Technology of China
dove.ustc@gmail.com

Hao Wang
360 Search Lab
cashenry@126.com

Zi Huang
School of Information Technology and Electrical Engineering, The University of Queensland
huang@itee.uq.edu.au

ABSTRACT

With the increasing popularity of various social media and E-commerce platforms, large volumes of user behaviour data (e.g., user transaction data, rating and review data) are being continually generated at unprecedented and ever-increasing scales. It is more realistic and practical to study recommender systems with inputs of streaming data. User-generated streaming data presents unique properties such as temporally ordered, continuous and high-velocity, which poses tremendous new challenges for the once very successful recommendation techniques. Although a few temporal or sequential recommender models have recently been developed based on recurrent neural models, most of them can only be applied to the session-based recommendation scenario, due to their short-term memories and the limited capability of capturing users' long-term stable interests. In this paper, we propose a streaming recommender model based on neural memory networks with external memories to capture and store both long-term stable interests and short-term dynamic interests in a unified way. An adaptive negative sampling framework based on Generative Adversarial Nets (GAN) is developed to optimize our proposed streaming recommender model, which effectively overcomes the limitations of classical negative sampling approaches and improves the effectiveness of the model parameter inference. Extensive experiments are conducted on two large-scale recommendation datasets, and the experimental results show the superiority of our proposed model in the streaming recommendation scenario.

ACM Reference Format:
Qinyong Wang, Hongzhi Yin, Zhiting Hu, Defu Lian, Hao Wang, and Zi Huang. 2018. Neural Memory Streaming Recommender Networks with Adversarial Training. In *Proceedings of The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3219819.3220004

*Corresponding author

1 INTRODUCTION

The recent study by Econsultancy [28] shows that 94% of companies agree that personalized recommendation has become an important means to improve user experience, engagement, revenue and conversion rate, and is critical to current and future success. For example, Amazon and Netflix recommendations are cited by many company observers as an outstanding means to improve revenues and profits. They rely on the ability of recommender systems to recommend media content or products that users are likely to consume. With the proliferation of smart mobile phones, mobile shopping and mobile commerce become the mainstream. In mobile commerce, most mobile purchases (more than 58%) are made after seeing a recommendation on a product detail page, and bringing the most personalized recommendations to the top of a small screen makes mobile shopping more convenient [34].

While recommender systems have attracted a large amount of research interest, the subarea of streaming recommender systems has remained largely unexplored. The real-world social media and E-commerce platforms such as Amazon and Netflix generate massive user-item interaction data (i.e., user behavior data) at an unprecedented rate. For example, nearly 1.5 billion transactions were made in Alibaba during its Singles Day shopping event on 11, November 2017 [30]. Such data is temporally ordered, continuous, high-speed and time-varying, which determines the streaming nature of data in recommendation systems. Therefore, it is more practical to study recommender systems using a streaming data framework. The real-time streaming setting poses great challenges for conventional recommender systems, most of which are designed for static settings and implemented by batch learning techniques, which usually suffer from the following drawbacks when dealing with the stream situation: 1) delay on model updates caused by the expensive time cost of re-running the batch model; 2) inability to track fast-changing trends (e.g., user preferences and item popularity) in the presence of streaming data; and 3) lots of memory overhead to explicitly store all the historical data.

As the recommender system by nature is an incremental process, some online learning techniques based on stochastic gradient descent (SGD) and particle filters [11, 54] have been adopted or developed to support streaming recommendations. However, they suffer from the problem of "short-term memory", i.e., since the algorithms to update these models are based only on the most recent data points, and do not take into account the past observations, the models quickly forget the patterns learned in earlier stages. The recommendation models based on these online learning algorithms fail to capture users' long-term stable interests. Recently, recurrent neural model based recommender systems [17, 49] are proposed to tackle streaming recommendation problems, which adopt recurrent neural networks (RNN) or long short-term memory (LSTM) to capture and store sequential patterns of interacted items or users' dynamic interests. Such models learn a continuous and compact representation of the input data, and perform global updates across the whole memory cell at each step. These architecture designs encourage RNNs/LSTMs to capture sequential patterns from data, which makes RNNs/LSTMs based recommender systems especially good at capturing sequential user activity patterns or user temporal dynamics. They are often applied to the session-based recommendation scenario without user profiles (i.e., users' historical interaction data). However, like the online learning algorithms, RNNs/LSTMs have limitations in modeling and capturing users' long-term stable interests which are hardly affected by temporal factors.

To this end, we propose a novel model named **N**eural **M**emory **R**ecommender **N**etworks (NMRN). NMRN is inspired by Neural Turing Machines (NTM) [14] and Memory Network (NemNN) [37], and consists of two main components: *augmented memories* storing user and item information, and an *controller network* interacting with user activity data and reading or writing to the memories. Augmenting controller network with external memories separates the tasks of storing and processing information and makes the network only focus on processing information stored outside it. It also increases its capability of storing knowledge, thus users' long-term and stable preferences can persist if necessary. Moreover, NMRN can flexibly and explicitly read from and write to the memories on demand like physical computers. NMRN encourages local changes in memory, and this ability to directly execute local update to internal memories makes NMRN powerful to integrate users' recent interaction data and capture their newly emerging interests.

Specifically, the external memories of NMRN compose the key memory and the value memory. The structure of key-value memory networks (KV-MemNN) [26, 57] makes the original MemNN more suitable to deal with pairwise data. Given a new user-item interaction pair (u, v) arriving at the system in real time, NMRN first generates a soft address from the key memory, activated by u. The addressing process is inspired by recent advances in attention mechanism, which is popular in the fields of computer vision [27, 51] and NLP [23, 59], but is rarely studied in the field of recommender systems. The atention mechanism applied in recommender systems is useful to improve the retrieval accuracy and model interpretability. The fundamental idea of our attention design is to learn a weighted representation across the key memory, which is converted into a probability distribution by the Softmax function as the soft address. Then, NMRN reads from the value memory based on the soft address, resulting in a vector that represents both long-term stable and short-term emerging interests of user u. Inspired by the success of pairwise personalized ranking models [32, 35] (e.g., BPR) in top-k recommendations, we adopt the Hinge-loss in our model optimization. As the number of unobserved examples is very huge, we employ the negative sampling method proposed in [24] to improve the training efficiency. Most existing negative sampling approaches use either random sampling [1, 24, 38] or popularity-biased sampling strategies [5, 11]. However, the majority of negative examples generated in these sampling strategies can be easily discriminated from observed examples, and will contribute little towards the training, because sampled items could be completely unrelated to the target user. Besides, these sampling approaches are not adaptive enough to generate adversarial negative examples, because (1) they are static and thus do not consider that the estimated similarity or proximity between a user and an item changes during the learning process. For example, the similarity between user u and a sampled noise item v is high at the beginning, but after several gradient descent steps it becomes low; and (2) these samplers are global and do not reflect how informative a noise item is w.r.t. a specific user. In light of this, we develop an adaptive noise sampler based on a Generative Adversarial Network (GAN) [13] to optimize our model, which considers both the specific user and the current values of the model parameters to adaptively generate "difficult" and informative negative examples. Moreover, in order to simultaneously capture the first-order similarity between users and items as well as the second-order similarities between users and between items to learn robust representations of users and items, we adopt the Euclidean distance to measure the similarity between a user and an item instead of the widely adopted dot product, inspired by [19].

Overall, we summarize the major contributions of this paper:

(1) To the best of our knowledge, we are the first to design and develop a streaming recommender model based on Key-Value Memory Networks to provide real-time recommendations. It has the nice capability of capturing and storing both users' long-term stable interests and short-term dynamic interests.
(2) We propose a novel adaptive noise sampler to generate adversarial negative samples for model optimization, which significantly improves training effectiveness of our proposed streaming recommender model. This is the first time to integrate Generative Adversarial Nets with the negative sampling method seamlessly for recommender systems.
(3) We conduct extensive experiments to evaluate the performance of our proposed streaming recommender model in the streaming recommendation setting on two large-scale recommendation datasets. The results show the superiority of our proposals by comparing with the state-of-the-art streaming recommendation techniques.

2 MODEL DESCRIPTION

In this section, we first formulate NMRN, then describe its structure and finally present how to use it for streaming recommendations. Note that in the description below, vectors and matrices are denoted with bold small letters and bold capital letters respectively.

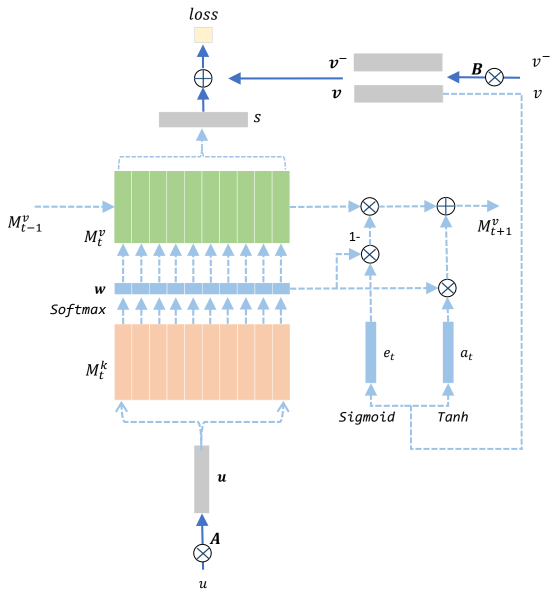


Figure 1: the Architecture of the Propose Model

2.1 Model Preliminaries

The detailed structure of NMRN is illustrated in Figure 1. The total numbers of users and items are denoted as N and M. We denote a user-item interaction pair at time t as (u_t, v_t) while v_t^- is a sampled negative item for a specific user at time t. Various sampling strategies have been developed to draw them, which will be described and analyzed in Section 3. u_t is transformed into $\mathbf{u_t}$ by multiplied with user embedding matrix $\mathbf{A}$ while v_t and v_t^- are transformed into $\mathbf{v_t}$ and $\mathbf{v_t^-}$ with item embedding matrix $\mathbf{B}$, where $\mathbf{u_t}$, $\mathbf{v_t}$ and $\mathbf{v_t^-}$ have real-valued vectors with r dimensions, and the size of $\mathbf{A}$ and $\mathbf{B}$ is $N \times r$. $\mathbf{M_t^k}$ and $\mathbf{M_t^v}$ respectively represent key memory matrix and value memory matrix at time t respectively, both with size of $L \times r$, where L and r stand for the number of memory slots and the dimension of each memory slot respectively. $\mathbf{w_t}$ is an attention weight vector with size L that is generated from $\mathbf{M_t^k}$ activated by a specific user u at time t. Thus, the weight vector is both user- and time-specific. Note that we leave out the subscript t of u_t, v_t, $\mathbf{u_t}$, $\mathbf{v_t}$, $\mathbf{v_t^-}$ and $\mathbf{w_t}$ to make the expressions more compact in the following sections and figures where no confusion occurs.

The notion of similarity is at the heart of all recommender models and neural memory models. We adopt the metric learning approach to learn a metric space, in which we employ the Euclidean distance to measure the similarity among users, items and memory slots. Given two data points $\mathbf{p}$ and $\mathbf{q}$ in a learned metric space, their Euclidean distance d is calculated as:

$$d(p, q) = \|\mathbf{p} - \mathbf{q}\| = \sqrt{\sum_i (p_i - q_i)^2} \tag{1}$$

Compared with the widely adopted dot product similarity, the distance metric meets the condition of the triangle inequality: for any three objects in a metric space, the sum of any two pairwise distance should be greater than or equal to the remaining pairwise distance. This property implies that: (1)if user u is close to both item v_1 and v_2, it implies that v_1 and v_2 are also close to each other bounded by $d(v_1, v_2) < d(u, v_1) + d(u, v_2)$; (2)similarly, if item v is close to both user u_1 and u_2, then u_1 and u_2 are also guaranteed to be close to each other bounded by $d(u_1, u_2) < d(u_1, v) + d(u_2, v)$ [19].

These implications could result in the following benefits in user behavior modelling: (1) The distance metric facilitate the following objects to cluster together: users who co-like the same items and items co-liked by the same users. (2) Given any user, his/her nearest neighboring items are the ones that he/she liked or are liked by other similar users.

2.2 Memory Addressing

We assume that there exist L latent factors for interests, and different user has different tastes (i.e., weights) on these latent factors. Each user's weights remain stable because they are related to his/her intrinsic personality factors. In our model, each key memory slot stores the key (i.e., weight) to the corresponding latent factor. Specifically, at time t, the one-hot representation of user u is transformed into an r-dimensional continuous user embedding $\mathbf{u}$ via the embedding matrix $\mathbf{A}$. Then $\mathbf{u}$ is fed into the key memory matrix $\mathbf{M_t^k}$ to get attention weights, i.e., $\mathbf{u}$ is compared with each key memory slot (take the i^{th} one as an example) to get the similarity sim_i using the Euclidean distance:

$$sim_i = \left\|\mathbf{u} - \mathbf{M_t^k}(i)\right\| \tag{2}$$

Then, an attention weight vector $\mathbf{w}$ with size L are obtained by applying the Softmax function, i.e., $Softmax(z_i) = e^{z_i}/\Sigma_j e^{(z_j)}$, to the negative similarities so that the most similar memory slot produces the largest weight:

$$w(i) = Softmax(-sim_i) \tag{3}$$

We adopt the soft attention mechanism by using the differentiable Softmax function so that the whole architecture is differentiable and end-to-end. This property is called end-to-end differentiable, with one end being the inputs and the other the outputs. The attention process is also referred as memory addressing to mimic real memory operations in the areas of computer architectures [26].

2.3 Memory Reading

Each slot in the value memory stores the concrete latent factor value (e.g., topics and categories) in the same order as the value memory. Unlike the stable weights, each of these latent factors' values is varying and evolutionary during the whole process in different speed, analogous to the topic evolution over time [29]. Significant change in latent factor value means that old preferences are disappearing while new preferences appear, i.e., new user interests emerge, but minor change in latent factor value means user interests remain stable. This explains why our model can capture both long-term stable interests and short-term dynamic interests. Technically, each memory slot is multiplied by the corresponding attention weight, resulting in a r-dimensional vector $\mathbf{s}$ which we call *proxy preference vector*:

$$\mathbf{s} = \sum_{i=1}^{L} w(i)\mathbf{M_t^v}(i) \tag{4}$$

The proxy preference vector $\mathbf{s}$ plays an important role and we explain it as follows: (1) It preserves the information of the user's all interacted items, so that it naturally captures the user's long-term and short-term interests. (2) It is generated from the value memory matrix that stores item information, therefore it could act as a proxy of a user when computing the similarity between a user and an item, because users and items may be projected to different spaces (e.g., when the size of a key memory slot is different from that of a value memory slot), and their similarity cannot be directly computed via Euclidean distance. In fact, the overall effect is equivalent to that the user and the item are projected into the joint r-dimensional space, which enables the similarity computation between a user and an item. Thus, given a user as query, the efficiency of the top-k recommendation at time t can be significantly improved with off-the-shelf approximate nearest-neighbor (ANN) algorithms, such as location-sensitive hashing (LSH).

2.4 Memory Writing

To locally update the value memory matrix to adapt to the change of user preferences, a memory writing scheme is proposed inspired by [14, 57]. Once a new user-item pair arrives at the system, the item's embedding will be written to the value memory matrix using the same attention weights $\mathbf{w}$ generated in the memory reading stage. The relevant memories are first erased and then the new item information such as the item's popularity is added to update the value memory.

Specifically, to update the value memory matrix $\mathbf{M}_\mathbf{t}^\mathbf{v}$ into $\mathbf{M}_{\mathbf{t+1}}^\mathbf{v}$, a linear function is applied to the newly arrived item v's embedding $\mathbf{v}$ and then the Sigmoid function is employed to the result, obtaining the *erased vector* $\mathbf{e_t}$ as follows:

$$\mathbf{e_t} = Sigmoid(\mathbf{W_e v} + \mathbf{b_e}) \tag{5}$$

where $\mathbf{W_e}$ is a linear transformation matrix with size of $r \times r$, and $\mathbf{b_e}$ is a r-dimensional bias vector. The resulting erased vector $\mathbf{e_t}$ is a r-dimensional vector, whose values range from 0 to 1. Then the value memory $\mathbf{M}_\mathbf{t}^\mathbf{v}$ is partially erased, and each of its memory slots is modified by:

$$\tilde{\mathbf{M}}_{\mathbf{t+1}}^\mathbf{v}(i) = \mathbf{M}_\mathbf{t}^\mathbf{v}(i) \circ [\mathbf{I} - w(i)\mathbf{e_t}] \tag{6}$$

where $\mathbf{I}$ is a r-dimensional vector with each element being 1, and $\circ$ is element-wise multiplication. Similar to the reading case, the weighting $w(i)$ tells us where to focus our erasing. In this way, the value location is reset to 0 if the corresponding weight and the erased vector element are both 1, and if either is 0, the value location remains unchanged. Following the erasing operation, a r-dimensional *add vector* $\mathbf{a_t}$ is calculated for updating the value memory in a similar manner:

$$\mathbf{a_t} = Tanh(\mathbf{W_a v} + \mathbf{b_a}) \tag{7}$$

where $\mathbf{W_a}$ and $\mathbf{b_a}$ are linear transformation matrix and bias vector with size $r \times r$ and r. Finally, the value memory matrix at time $t+1$ is updated to be $\mathbf{M}_{\mathbf{t+1}}^\mathbf{v}(i)$ as follows:

$$\mathbf{M}_{\mathbf{t+1}}^\mathbf{v}(i) = \tilde{\mathbf{M}}_{\mathbf{t+1}}^\mathbf{v}(i) + w(i)\mathbf{a_t} \tag{8}$$

2.5 Pairwise Loss Function

Inspired by [19, 22, 32], we design a pairwise loss function based on Hinge-loss. Our intuition is that a user's proxy preference vector $\mathbf{s}$ should be similar with the interacted items while dissimilar with the non-interacted items in order to act as a proper proxy between users and items. Therefore, the loss function should pull the positive items (i.e., neighbors) closer and push the negative items (i.e., impostors) further from the proxy preference vector. As the number of non-interacted items of each user is huge, we adopt the negative sampling method to perform model optimization instead of using all unobserved examples, following skip-gram model [25]. The pairwise loss function is defined as follows:

$$\mathcal{L} = \sum\nolimits_{(u,v)\in\mathcal{S}, v^- \sim \mathcal{V}_u^-} w_{u,v} * [m + d(u,v) - d(u,v^-)]_+ \tag{9}$$

where $[z]_+ = max(z; 0)$ denotes the standard Hinge-loss, $w_{u,v}$ is the ranking loss weight (described later) and $m > 0$ is the safety margin size; $\mathcal{S}$ is the user activities data currently available for training, $\mathcal{V}_u^-$ is a set of items that u has never interacted with, currently, we simply assume that each negative item v^- is uniformly drawn from $\mathcal{V}_u^-$, but we will introduce a better-designed adaptive negative sampling method based on GAN in Section 3; $d(u,v)/d(u,v^-)$ is the distance between u's proxy preference vector $\mathbf{s}$ and the positive/negative item vector $\mathbf{v}/\mathbf{v}^-$.

2.6 Ranking Loss Weights and Regularization

A rank-based weighting scheme called Weighted Approximate-Rank Pairwise (WARP) loss [46, 47] is adopted to penalize items at a lower rank. Given a user u, let $rank_{u,v}$ denote the rank of item v in u's recommendation list, we penalize a positive item v based on its rank by setting:

$$w_{u,v} = \log(rank_{u,v} + 1) \tag{10}$$

This scheme penalizes a positive item at a lower rank much more heavily than at the top. However, it is computationally inefficient to rank all items for each user. To avoid ranking all items, we adopt the following approximate method to obtain $rank_{u,v}$: for each positive item v of user u, it needs to draw $N_{u,v}$ negative items until a v^- satisfying $d(u,v) - d(u,v^-) + m > 0$, i.e., an impostor, is found. Then the approximate rank is calculated as:

$$rank_{u,v} \approx \lfloor \frac{N-1}{N_{u,v}} \rfloor \tag{11}$$

Recall that N is the total number of items.

If the data points in a high-dimensional space spread too widely, the model would fail due to the curse of dimensionality [12]. To solve it, we bound the norm of all user/item vectors within 1 to ensure the model robustness [19]: $\|\mathbf{u}\| \leq 1$ and $\|\mathbf{v}\| \leq 1$.

2.7 Top-k Recommendations with NMRN

Generating top-k recommendations for a online user u at time t is straightforward with NMRN: u is transformed into her proxy preference vector $\mathbf{s}$ by embedding matrix $\mathbf{A}$, key memory $\mathbf{M}_\mathbf{t}^\mathbf{k}$ and value memory $\mathbf{M}_\mathbf{t}^\mathbf{v}$. Then $\mathbf{s}$ is compared with all items vectors transformed by embedding matrix $\mathbf{B}$ to find out the most similar k items according to $d(s,v)$. Due to the nice property of Euclidean distance, NMRN has the advantage of significantly speeding up the retrieval process with approximate nearest-neighbor (ANN) algorithms, such as location-sensitive hashing (LSH), especially when a well-known industrial LSH library *Annoy* is adopted. After that, the model gets updated based on this new item with the writing operation.

3 ADVERSARIAL TRAINING FRAMEWORK

In this section, we propose a GAN-based adversarial training framework to overcome the drawbacks of existing sampling methods.

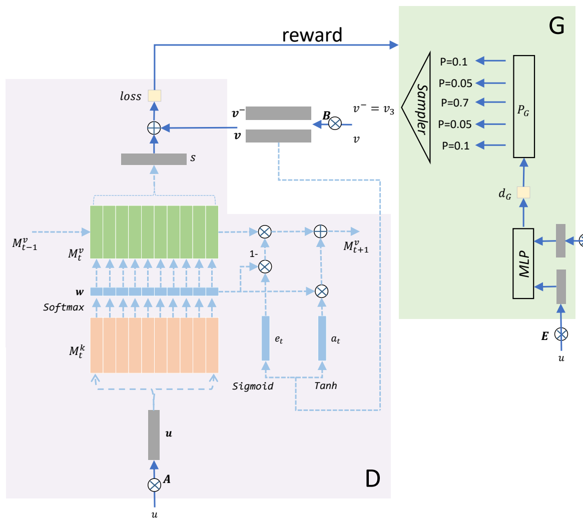


Figure 2: Overview of the Adversarial Training Framework

3.1 Problems of Existing Sampling Methods

To obtain a negative item, the most straightforward and widely adopted method is random sampling [1, 7, 8, 24, 38] or popularity-biased sampling strategies [5, 11]. However, there exist some problems that make them unsuitable for our application. For random sampling methods, the major problem is that the randomly generated items create too many "easy" tasks for the model to learn, i.e., these items might be completely unrelated to the user, so the model could easily discriminate them from the observed ones. These easy tasks contribute very little for the model optimization and hinder the model to discover more complex hidden user/item representations. On the contrary, a good and "difficult" item should be informative and could confuse the model if it has not captured the deep user/item representations. Moreover, the common problem for both methods is that they are not adaptive enough to generate adversarial negative items, i.e., they are static and do not consider the change of similarity or proximity between a user and an item during training. For example, before the training, the item v is sampled as noise because its similarity with user u is high, however, after several optimization iterations, the degree of similarity might drop, this item v with little information is still considered as noise. Another common problem is that the sampling probability distribution is usually global that applies to every user, but different users have different interests, thus that global distribution dose not reflect how informative a noise item is w.r.t a specific user. To this end, we propose an adversarial training framework to adaptively generate "difficult" and informative negative examples, highly inspired by KBGAN [2].

3.2 Framework Description

The proposed training framework is shown in Figure 2. In parallel to the GAN literature, we name the two models in the proposed framework as *Discriminator* and *Generator* denoted by D and G respectively. The target for the discriminator is to tell apart the true items from false items produced by the generator while for the generator is to adaptively generate an adversarial negative item and use it to deceive the discriminator. In the adversarial training settings, the generator and the discriminator are trained with respective to its own loss function.

3.2.1 *Discriminator.*

For the discriminator in our framework, its loss function is:

$$\mathcal{L}_D = \sum_{(u,v)\in\mathcal{S}} w_{u,v} * [m + d_D(u,v) - d_D(u,v^-)]_+ \qquad (12)$$
$$(u,v^-) \sim P_G(u,v^-|u,v)$$

where $P_G(u,v^-|u,v)$ is the probability distribution for generating negative tuple (u,v^-) by the generator given a positive tuple (u,v) (defined below); d_D denotes the distance measured by the discriminator, which is identical to d defined in in Eq (9). The optimization procedures follow what we described above. It is clear that the only difference between Eq (9) and Eq (12) is that the negative item v^- in Eq (12) is generated by the generator while it is drawn randomly in Eq (9).

3.2.2 *Generator.*

The objective of the generator maximizes the expectation of $-d_D(u,v^-)$ using the generated items so as to encourage the generator to generate plausible items to confuse the discriminator:

$$\mathcal{L}_G = \sum_{(u,v)\in\mathcal{S}} \mathbb{E}[-d_D(u,v^-)] \qquad (13)$$
$$(u,v^-) \sim P_G(u,v^-|u,v)$$

The distribution $P_G(u,v^-|u,v)$ is modeled as:

$$P_G(u,v^-|u,v) = \frac{exp(-d_G(u,v^-))}{\sum_{\bar{v}\in\mathcal{V}_u^-} exp(-d_G(u,\bar{v}))} \qquad (14)$$

where d_G is the Euclidean distance between user u and item v measured by the generator though a neural network. We embed u with matrix **E** and embed v with matrix **F**, the resulting vectors are separately forward-propagated by a multilayer perceptron (MLP) such that user u and item v are in the same space, and d_G is measured between the output user and item vectors, as shown in Figure 2. $\mathcal{V}_u^-$ is a set containing all items not interacted with user u in the history, however, this set might be large, witch would significantly damage the efficiency. In order to reduce computational complexity, we only uniformly randomly select T items from the original whole $\mathcal{V}_u^-$ to form a new but smaller $\mathcal{V}_u^-$.

However, the challenge in the optimization process is that $\mathcal{L}_G$ can not be directly optimized by SGD because it involves a discrete sampling step which blocks the flow of gradients. Following the work [2, 56], we consider it as a reinforcement learning problem, where, analogously, (u,v) is the *state*, $P_G(u,v^-|u,v)$ is the *policy*, (u,v^-) is the *action* and $-d_D(u,v^-)$ is the *reward*.

A common optimization approach is to adopt policy gradient based reinforcement learning (REINFORCE) [48, 56]. According to that, the gradient of $\mathcal{L}_G$ with respect to its parameters is:

$$\begin{aligned}\nabla_{\theta_G}\mathcal{L}_G &= \sum_{(u,v)\in\mathcal{S}} \mathbb{E}_{(u,v^-)\sim P_G}[-d_D(u,v^-)\nabla_{\theta_G} logP_G(u,v^-|u,v)] \\ &\simeq \sum_{(u,v)\in\mathcal{S}} \frac{1}{T} \sum_{(u_i,v_i^-)\sim P_G,\, i=1}^{T} [-d_D(u_i,v_i^-)\nabla_{\theta_G} logP_G(u_i,v_i^-|u,v)]\end{aligned} \qquad (15)$$

REINFORCE algorithm often suffers from the issue of high variance. A widely adopted solution is subtracting a baseline b from the policy gradient. Although a good baseline should be the function of the state value [31], it could also be chosen arbitrarily without

Algorithm 1: The Adversarial Training Algorithm

```
1  INPUT: user-item interaction history $(u,v) \in \mathcal{S}$, margin $m$ and learning rates $\lambda_D$ and $\lambda_G$;
2  OUTPUT: the discriminator $D$, the generator $G$ and their distance measurement function $d_D$ and $d_G$;
3  Initialize the parameters $\theta_D$ and $\theta_G$ for $D$ and $G$;
4  $b = 0$; //initiate the baseline
5  Pre-train $D$ and $G$ w.r.t. to $\theta_D$ and $\theta_G$;
6  while not convergent do
7      Sample a mini-batch $\mathcal{S}_{batch} \in \mathcal{S}$; $\Delta_G = 0$; $\Delta_D = 0$; $R_{batch} = 0$;//zero the gradients of $D$ and $G$ and rewards
8      for $(u,v) \in \mathcal{S}_{batch}$ do
9          Compute its ranking weight $w_{(u,v)}$;
10         Uniformly randomly sample $T$ negative items: $\mathcal{V}_u^-$;
11         Measure the sampling probability distribution: $P_G(u,v^-|u,v) = \frac{exp(-d_G(u,v^-))}{\sum_{\bar{v} \in \mathcal{V}_u^-} exp(-d_G(u,\bar{v}))}$;
12         Sample a negative item $v^-$ forming $(u,v^-)$ by the distribution $P_G(u,v^-|u,v)$;
13         Freeze the weights of $G$;
14         $\Delta_D = \Delta_D + \nabla_{\theta_D}\{w_{u,v} * [m + d_D(u,v) - d_D(u,v^-)]_+\}$; // accumulate the example gradient to $\Delta_D$
15         $R_{batch} = R_{batch} + (-\mathcal{D}(u,v^-))$;
16         Freeze the weights of $D$;
17         $\Delta_G = \Delta_G + (R_{batch} - b)\nabla_{\theta_G} \log p_G$; // accumulate the example gradient to $\Delta_G$
18     end
19     $\theta_D = \theta_D - \lambda_D \Delta_D$;$\theta_G = \theta_G + \lambda_G \Delta_G$; //minimize the loss of $D$ while maximize the likelihood of $G$
20     $b = \frac{R_{batch}}{|\mathcal{S}_{batch}|}$; //update the baseline
21 end
```

changing the expectation to avoid introducing new parameters [48]. In our case, we fix keep b fix as the average reward of the whole training set, i.e., $b = \sum_{(u,v)\in\mathcal{S}} \mathbb{E}_{(u,v^-)\sim P_G(u,v^-|u,v)}[-d_D(u,v^-)]$. For implementation, only recent generated negative items are used to approximate b.

Note that like many typical GAN models [40, 58], both the discriminator and the generator are pre-trained separately. In particular, the discriminator is pre-trained according to Eq (9) with random negative items and the generator is pre-trained by maximizing the log-likelihood of Eq (14). Algorithm 1 summaries this adversarial training process.

4 EVALUATION

In this section, we first introduce the experimental settings and then report the experimental results.

4.1 Datasets

To evaluate our streaming recommendation model, we select two large-scale and publicly available datasets, i.e., Movielens and Netflix, that contain time information of user-item interactions in order to simulate the real streaming scenario.

Movielens: Movielens is a widely adopted movie dataset for evaluating recommender systems. We choose the *20M Dataset* that contains *20 million* interaction records generated by *138493* users on *26744* movies. All interaction records are associated with timestamps ranging from the *01/09/1995* to *03/31/2015*.

Netflix: Netflix is another movie dataset that consists of *24 million* interaction records, *470758* users and *4499* movies. This dataset was sampled between November, 1999 and December, 2005 and reflects the distribution of all interactions received during this period, and the temporal granularity is a day..

4.2 Baseline Methods

We compare our proposed models with three state-of-the-art recommendation models that support online updating. To validate the effect of our proposed GAN-based adaptive negative sampling approach (called NMRN-GAN for the rest of this paper), we design a baseline called NMRN-RS that adopts the simple non-adaptive random sampling method to draw negative items.

RRN Recurrent Recommender Networks (RRN) [49] are able to predict future behavioral trajectories. This is achieved by endowing both users and movies with a Long Short-Term Memory (LSTM) [18] autoregressive model that captures temporal dynamics of user interests, in addition to a more traditional low-rank factorization to capture users' stable interests.

RKMF Regularized Kernel Matrix Factorization (RKMF) is proposed by Rendle et. al in [33], inspired by that the nonlinear interactions between feature vectors are possible with kernels. A flexible online learning algorithm is developed for RKMF to update on selected data instances. We use a RKMF implemented by *mfrec* [1].

WARP Weighted Approximate-Rank Pairwise (WARP) [46] loss is the state-of-the-art top-k recommendation method specifically designed for implicit feedbacks. WARP uses SGD and a smart sampling trick to approximate ranks, resulting in an efficient online optimization strategy. A WARP implemented by *LightFM*[2] is used.

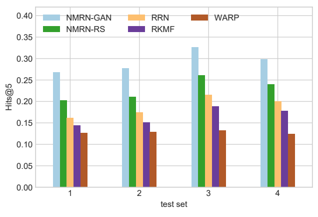
NMRN-RS NMRN-RS is a NMRN implementation that only uses uniformly random sampling strategy to draw negative items. We compare with it to validate the benefits brought by our proposed GAN-based adaptive negative sampling approach.

4.3 Evaluation Setup

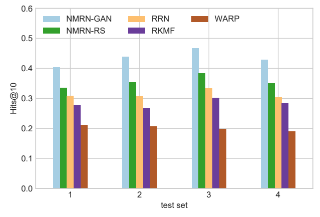
In this section, we describe how to simulate real streaming recommendation scenarios and the evaluation protocol and measurement.

4.3.1 Scenario Simulation. To mimic a real streaming recommendation scenario, we need to split the datasets properly. Besides online streaming recommendation, we also test our model in a traditional offline batch-based temporal recommendation setting [52].

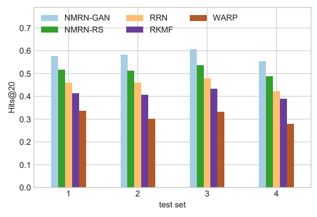
[1]github.com/mlaprise/mfrec
[2]github.com/lyst/lightfm



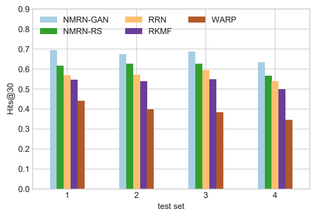
(A) *Hits@5*



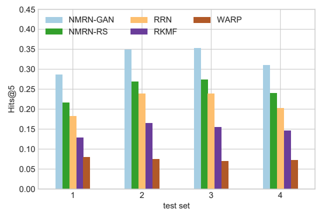
(B) *Hits@10*



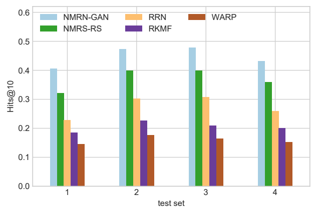
(C) *Hits@20*



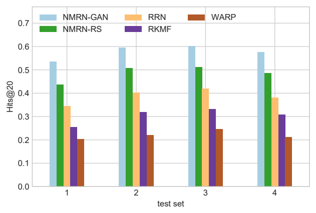
(D) *Hits@30*



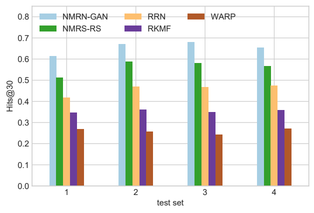
(E) *Hits@5*



(F) *Hits@10*



(G) *Hits@20*



(H) *Hits@30*

Figure 3: *Hits@k* on Movielens (A-D) and Netflix (E-H) with Increasing *k* for Online Streaming Setting

For online streaming recommendation, we follow the data splitting strategy introduced in [3]. We order all interaction records by timestamp and evenly split them into 6 parts. The first 2 parts form a base training set are denoted as $\mathcal{D}^{train}$ and the remaining 4 parts are test sets denoted as $\mathcal{D}_1^{test} \dots \mathcal{D}_4^{test}$, which are used to mimic the online streaming inputs. All the comparison methods are first trained over $\mathcal{D}^{train}$ to determine the initial model parameters and the best hyper parameters (e.g., the number of latent factors and the shapes of embedding). Once the initial training is completed, the 4 test sets are sequentially predicted. During testing, the previous test set D_i^{test} is used to update the model, then the updated model predicts the current test set D_{i+1}^{test}. Formally, the model is first trained with $\mathcal{D}^{train}$, resulting in $\mathcal{M}_0$, then $\mathcal{M}_0$ is used to perform evaluation on $\mathcal{D}_1^{test}$. After that, $\mathcal{M}_0$ is updated with $\mathcal{D}_1^{test}$ to be $\mathcal{M}_1$ and so on. For *new users* not contained in the training set, we recommend the most popular items.

For offline batch recommendation, we order all user interaction records in dataset $\mathcal{D}$ by their timestamps and split them into halves, denoted as $\mathcal{D}^{train}$ and $\mathcal{D}^{test}$. $\mathcal{D}^{train}$ is used to learn model parameters and $\mathcal{D}^{test}$ is for evaluating trained model.

4.3.2 Evaluation Methodology. The evaluation methodology we adopt for all comparison methods is *Hits@k* that is widely applied in the recommendation research area [6, 20, 43]. For each user interaction record (u, v) with time t in the test set: 1) We randomly sample $\mathcal{J}$ items that user u has never interacted with before t. It is worth mentioning that $\mathcal{J}$ is set to 5000 and 500 for the Movielens and Netflix respectively in our experiments; 2) Compute u's preference scores for these $\mathcal{J}$ items and v; 3) Sort these $\mathcal{J}$*+1* items by their scores to form a ranked list. Let p denote the position of v within the list and thus the best result happens when v precedes all the other items (i.e, p = 1); 4) A top-k recommendation list is formed by picking the k items with highest scores. If $p \leq k$, we get a *hit* (i.e., the ground truth v appears in u's recommendation list), otherwise, we get a *miss*. The hit probability would rise as k increases or $\mathcal{J}$ decreases, thus we need to fix k and $\mathcal{J}$ for all comparison methods to ensure fairness. 4) The *Hit@k* is finally computed as:

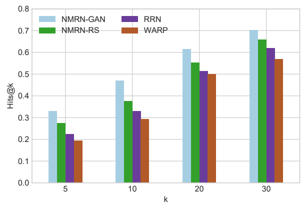
$$Hits@k = \frac{\#hit@k}{|\mathcal{D}^{test}|} \tag{16}$$

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

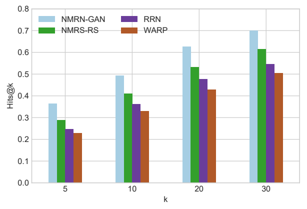
4.4 Experimental Results

In this section, we present the experimental results in both online streaming and offline batch recommendation settings, and the optimal hyper-parameters are also given for the convenience of repeating our experiments.

Our proposed NMRN-GAN achieves its best performance with the hyper-parameters r=64, L=128, T=200 and m=3 on the Movielens dataset, and r=32, L=64, T=100 and m=4 for the Netflix dataset. Figure 3 reports the online streaming recommendation accuracy in terms of *Hits@5*, *Hits@10*, *Hits@20* and *Hits@30* on both datasets. The horizontal axis is the timeline, four reference time points are chosen to simulate the "the current time". Obviously, our proposed NMRN models (including NMRN-GAN and NMRN-RS) significantly outperform the other comparison methods on both datasets. In addition, several other observations are made from the results. (1) The LSTM based recommender RRN performs much better than other comparison methods WARP and RKMF. which validates our statement that a streaming recommender system should preserve long-term stable personal interests as well as capture the short-term newly emerging interests. (2) However, RRN falls far behind our NMRS models, although it also considers both long-term stable and short-term dynamic user interests. This is because RRN uses two separate models (i.e., LSTM and the traditional matrix factorization) to model users' two types of interests respectively and then simply adds the results of these two models together to produce the final recommendations. In contrast, our NMRS models with external key-value memories naturally capture and store both long-term stable and newly emerging interests in a unified manner. (3) NMRN-GAN



(A) Movielens



(B) Netflix

Figure 4: Batch Recommendation on Movielens and Netflix

performs better than NMRN-RS, which demonstrates the significance of adaptively generating adversarial negative items to, in turn, improve the discriminator. (4) Although the *Hits@k* values of models on the Movielens dataset look similar to that achieved on the Netflix dataset, in fact all of them perform worse on the Netflix dataset, since a much smaller $\mathcal{J}$ is adopted for the Nexflix dataset to ensure that *Hits@k* values in both datasets are on the same scale (refer to Section 4.3.2). This is because the Netflix dataset is a sample dataset, and the average number of movies associated with each user is much smaller than that on the Movielens dataset. This sparsity issue is exacerbated and augmented in the streaming recommendation scenario.

Since WARP and RRN are not originally proposed for streaming recommendation, we also perform evaluations under the offline batch recommendation setting described in Section 4.3.1. We report the results in Figure 4, and find that NMRN-GAN still beats other competitors in the traditional top-k recommendation task, validating that NMRN-GAN is capable of preserving and storing users' long-term stable interests. We also observe that (1) WARP in this setting performs much better than in the streaming recommendation setting; (2) All models achieve higher *Hits@k* recommendation accuracy values in this recommendation setting, and actually their performances in this setting are an upper bound for their performance in the online streaming recommendation scenario.

5 RELATED WORK

5.1 Online Recommender Systems

Temporal information is a significant factor to consider when designing personalized recommender systems [39, 41, 42, 45, 50, 52, 53, 55]. Streaming recommendation treats temporal information in a temporally ordered, continuous and high-velocity setting. There are two main categories of existing streaming recommender systems by the way they get online updated.

The first category is neighborhood-based streaming recommender systems. These recommendation methods heavily rely on an offline phase where typically a neighborhood-based method is employed to calculate the similarity among users, which is used to make recommendations for the user based on the current preferences of his/her most similar users obtained in the past [9, 21, 36]. [9] proposed online models to generate personalized recommendations for Google News users. [21] leveraged the considerable computation ability of Storm, a data access component and a specially developed data storage component. The largest drawback for these models is that every update requires re-computing all similarities for the offline model, and clearly, the retraining process makes it extremely inefficient for the streaming recommendation task. Another problem is that they assume all historical user interaction records can fit into the main memory, which is not always possible. Our proposed model cost much less memory usage because it dose not need to store the whole historical interaction records, rather, they are abstracted into key memory and value memory.

Another category is model-based streaming recommender systems, which incrementally update a trained model based on user interactions arriving in the temporal order constantly, thus the model is expected to get updated for every one or batch of new user interaction(s). Various online update methods are designed for these models [4, 10, 11, 16, 44]. [10, 16] focused on improving the online recommendation with implicit feedbacks and they paid more attention to the negative sampling in the streaming settings. [11] assumed that the system cannot deal with all input data in streaming setting and they update the model only based on the sampled data maintained in a well-designed reservoir. A flexible online-update algorithm is developed for RKMF model [33] which updates the model on selected new data instances.

5.2 Neural Memory Networks

The neural memory network (MemNN) is a concept inspired by computer architectures and it has strong abilities to capture long-term dependencies [14]. MemNN saw its success in many applications such as question answering [26, 37], neural language translation [15] and knowledge tracking [57] but they are rarely studied in the field of recommendation systems, especially in streaming settings. Typically, a MemNN model consists of two parts, a memory to store the information and a controller to interact with data, in particular, to read from and write to the memory. Our model is based specifically on Key-value paired memory networks, which are a generalization of the way contexts (e.g. knowledge bases or documents to be read) are stored in memory. The lookup (addressing) stage is based on the key memory while the reading stage (given the returned address) uses the value memory. This provides more flexibility and effectiveness [26].

MemNN that our model is based on can be considered as a generalization of traditional RNN/LSTM and they are different in two main aspects. First, MemNN encourages local changes in memory, which can discover not only the structures in the training data but can also generalize to structures beyond that. The other difference is that MemNN has larger external memories compared to RNN/LSTM that use a single hidden state vector to store information.

6 CONCLUSIONS

User activity data arrive at real-world recommender systems rapidly and continuously, which determines the use of recommender systems in streaming settings. In this paper, we proposed a novel streaming recommender model based on the key-value memory networks to capture and store both long-term stable interests and short-term dynamic interests in a unified way. To improve the effectiveness of model training, we designed an adaptive negative sampling scheme based on GAN to generate adversarial negative samples for each user. To validate our statements, we conducted extensive experiments to evaluate the performance of our proposed streaming recommender models qualitatively and quantitatively. The experimental results showed that our streaming recommender models significantly outperform some strong and state-of-the-art online or streaming recommender models.

7 ACKNOWLEDGEMENT

This work was supported by ARC Discovery Early Career Researcher Award (Grant No. DE160100308), ARC Discovery Project (Grant No. DP170103954) and New Staff Research Grant of The University of Queensland (Grant No. 613134). It was also partially supported by National Natural Science Foundation of China (Grant No. 61572335 and No. 61628206).

REFERENCES

[1] Trapit Bansal, David Belanger, and Andrew McCallum. 2016. Ask the Gru: Multi-Task Learning for Deep Text Recommendations. In *RecSys*. 107–114.

[2] Liwei Cai and William Yang Wang. 2017. KBGAN: Adversarial Learning for Knowledge Graph Embeddings. *arXiv* (2017).

[3] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A Hasegawa-Johnson, and Thomas S Huang. 2017. Streaming Recommender Systems. In *WWW*. 381–389.

[4] Chen Chen, Hongzhi Yin, Junjie Yao, and Bin Cui. 2013. Terec: A Temporal Recommender System over Tweet Stream. *VLDB* 6, 12 (2013), 1254–1257.

[5] Ting Chen, Yizhou Sun, Yue Shi, and Liangjie Hong. 2017. On Sampling Strategies for Neural Network-based Collaborative Filtering. In *SIGKDD*. 767–776.

[6] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of Recommender Algorithms on Top-N Recommendation Tasks. In *RecSys*. 39–46.

[7] Peng Cui, Shaowei Liu, and Wenwu Zhu. 2018. General Knowledge Embedded Image Representation Learning. *IEEE Transactions on Multimedia* 20, 1 (2018), 198–207.

[8] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2017. A Survey on Network Embedding. *arXiv* (2017).

[9] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google News Personalization: Scalable Online Collaborative Filtering. In *WWW*. 271–280.

[10] Robin Devooght, Nicolas Kourtellis, and Amin Mantrach. 2015. Dynamic Matrix Factorization with Priors on Unknown Values. In *SIGKDD*. 189–198.

[11] Ernesto Diaz-Aviles, Lucas Drumond, Lars Schmidt-Thieme, and Wolfgang Nejdl. 2012. Real-Time Top-n Recommendation in Social Streams. In *RecSys*. 59–66.

[12] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. Springer.

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

[14] Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural Turing Machines. *arXiv* (2014).

[15] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to Transduce with Unbounded Memory. In *NIPS*. 1828–1836.

[16] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast Matrix Factorization for Online Recommendation with Implicit Feedback. In *SiGIR*. 549–558.

[17] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-Based Recommendations with Recurrent Neural Networks. *arXiv* (2015).

[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

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

[20] Bo Hu and Martin Ester. 2013. Spatial Topic Modeling in Online Social Media for Location Recommendation. In *RecSys*. 25–32.

[21] Yanxiang Huang, Bin Cui, Wenyu Zhang, Jie Jiang, and Ying Xu. 2015. Tencentrec: Real-Time Stream Recommendation in Practice. In *SIGMOD*. 227–238.

[22] Yehuda Koren. 2008. Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model. In *SIGKDD*. 426–434.

[23] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective Approaches to Attention-Based Neural Machine Translation. *arXiv* (2015).

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv* (2013).

[25] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *NIPS*. 3111–3119.

[26] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value Memory Networks for Directly Reading Documents. *arXiv* (2016).

[27] Volodymyr Mnih, Nicolas Heess, Alex Graves, and others. 2014. Recurrent Models of Visual Attention. In *NIPS*. 2204–2212.

[28] David Moth. 2013. 94% of businesses say personalisation is critical to their success. (2013). https://econsultancy.com/blog/62583-94-of-businesses-say-personalisation-is-critical-to-their-success

[29] Subhabrata Mukherjee, Hemank Lamba, and Gerhard Weikum. 2017. Item Recommendation with Evolving User Preferences and Experience. *arXiv* (2017).

[30] Grace Noto. 2017. Alipay Dominates Alibaba Singles Day With 90% of Transactions. (2017). https://bankinnovation.net/2017/11/alipay-dominates-alibaba-singles-day-with-90-of-transactions/

[31] Jan Peters and Stefan Schaal. 2008. Reinforcement Learning of Motor Skills with Policy Gradients. *Neural Networks* 21, 4 (2008), 682–697.

[32] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI*. 452–461.

[33] Steffen Rendle and Lars Schmidt-Thieme. 2008. Online-Updating Regularized Kernel Matrix Factorization Models for Large-Scale Recommender Systems. In *RecSys*. 251–258.

[34] GTanmay Seth. 2017. M-Commerce Trends To Watch Out For In 2017. (2017). https://www.knowarth.com/m-commerce-trends-to-watch-out-for-in-2017/

[35] Amit Sharma and Baoshi Yan. 2013. Pairwise Learning in Recommendation: Experiments with Community Recommendation on Linkedin. In *RecSys*. 193–200.

[36] Karthik Subbian, Charu Aggarwal, and Kshiteesh Hegde. 2016. Recommendations for Streaming Data. In *CIKM*. 2185–2190.

[37] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, and others. 2015. End-To-End Memory Networks. In *NIPS*. 2440–2448.

[38] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-Scale Information Network Embedding. In *WWW*. 1067–1077.

[39] Hao Wang, Yanmei Fu, Qinyong Wang, Hongzhi Yin, Changying Du, and Hui Xiong. 2017. A Location-Sentiment-Aware Recommender System for Both Home-Town and Out-of-Town Users. In *KDD*. 1135–1143.

[40] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *SIGIR*. ACM, 515–524.

[41] Sibo Wang and Yufei Tao. 2018. Efficient Algorithms for Finding Approximate Heavy Hitters in Personalized PageRanks. In *SIGMOD*. ACM.

[42] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *SIGKDD*. ACM, 505–514.

[43] Weiqing Wang, Hongzhi Yin, Ling Chen, Yizhou Sun, Shazia Sadiq, and Xiaofang Zhou. 2015. Geo-Sage: A Geographical Sparse Additive Generative Model for Spatial Item Recommendation. In *SIGKDD*. 1255–1264.

[44] Weiqing Wang, Hongzhi Yin, Qinyong Huang, Zi Wang, Xingzhong Du, and Nguyen Quoc Viet Hung. 2018. Streaming Ranking Based Recommender Systems. In *SIGIR*.

[45] Weiqing Wang, Hongzhi Yin, Shazia Sadiq, Ling Chen, Min Xie, and Xiaofang Zhou. 2016. SPORE: A Sequential Personalized Spatial Item Recommender System. In *ICDE*. 954–965.

[46] Jason Weston, Samy Bengio, and Nicolas Usunier. 2011. Wsabie: Scaling Up to Large Vocabulary Image Annotation. In *IJCAI*, Vol. 11. 2764–2770.

[47] Jason Weston, Hector Yee, and Ron J Weiss. 2013. Learning to Rank Recommendations with the K-Order Statistic Loss. In *RecSys*. 245–248.

[48] Ronald J Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* 8, 3-4 (1992), 229–256.

[49] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent Recommender Networks. In *WSDM*. 495–503.

[50] Min Xie, Hongzhi Yin, Hao Wang, Fanjiang Xu, Weitong Chen, and Sen Wang. 2016. Learning Graph-based POI Embedding for Location-Based Recommendation. In *CIKM*. 15–24.

[51] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, Attend And Tell: Neural Image Caption Generation with Visual Attention. In *ICML*. 2048–2057.

[52] Hongzhi Yin, Bin Cui, Ling Chen, Zhiting Hu, and Zi Huang. 2014. A Temporal Context-Aware Model for User Behavior Modeling in Social Media Systems. In *SIGMOD*. 1543–1554.

[53] Hongzhi Yin, Bin Cui, Ling Chen, Zhiting Hu, and Xiaofang Zhou. 2015. Dynamic User Modeling in Social Media Systems. *TOIS* 33, 3 (2015), 10.

[54] Hongzhi Yin, Bin Cui, Xiaofang Zhou, Weiqing Wang, Zi Huang, and Shazia Sadiq. 2016. Joint Modeling of User Check-In Behaviors for Real-Time Point-Of-Interest Recommendation. *TOIS* 35, 2 (2016), 11.

[55] Hongzhi Yin, Weiqing Wang, Hao Wang, Ling Chen, and Xiaofang Zhou. 2017. Spatial-Aware Hierarchical Collaborative Deep Learning for POI Recommendation. *TKDE* 29, 11 (2017), 2537–2551.

[56] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient.. In *AAAI*. 2852–2858.

[57] Jiani Zhang, Xingjian Shi, Irwin King, and Dit-Yan Yeung. 2017. Dynamic Key-Value Memory Networks for Knowledge Tracing. In *WWW*. 765–774.

[58] Yizhe Zhang, Zhe Gan, and Lawrence Carin. 2016. Generating Text via Adversarial Training. In *NIPS workshop on Adversarial Training*.

[59] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. 2016. Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification. In *ACL*, Vol. 2. 207–212.