Interactive Recommendation via Deep Neural Memory Augmented Contextual Bandits

Yilin Shen, Yue Deng, Avik Ray, Hongxia Jin
Samsung Research America, Mountain View, CA, USA
{yilin.shen,y1.deng,avik.r,hongxia.jin}@samsung.com

ABSTRACT

Personalized recommendation with user interactions has become increasingly popular nowadays in many applications with dynamic change of contents (news, media, etc.). Existing approaches model user interactive recommendation as a contextual bandit problem to balance the trade-off between exploration and exploitation. However, these solutions require a large number of interactions with each user to provide high quality personalized recommendations. To mitigate this limitation, we design a novel deep neural memory augmented mechanism to model and track the history state for each user based on his previous interactions. As such, the user's preferences on new items can be quickly learned within a small number of interactions. Moreover, we develop new algorithms to leverage large amount of all users' history data for offline model training and online model fine tuning for each user with the focus of policy evaluation. Extensive experiments on different synthetic and real-world datasets validate that our proposed approach consistently outperforms a variety of state-of-the-art approaches.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; *Online learning settings*; *Neural networks*; *Reinforcement learning*;

KEYWORDS

Interactive Recommendation, Contextual Bandits, Deep Learning, Online Learning

ACM Reference Format:
Yilin Shen, Yue Deng, Avik Ray, Hongxia Jin. 2018. Interactive Recommendation via Deep Neural Memory Augmented Contextual Bandits. In *Twelfth ACM Conference on Recommender Systems (RecSys '18), October 2–7, 2018, Vancouver, BC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3240323.3240344

1 INTRODUCTION

Personalized recommender systems have become the key business drive of various services, such as artificially intelligent personal assistants, smart TVs, online web services and so on. Most service providers usually require maintaining a large amount of contents and collecting user history data to provide a list of recommendations. However, these methods are not applicable to a majority of practical scenarios due to the dynamic change of contents, e.g., current news, new products, etc. Thus, it is highly desirable to quickly adapt to user's preferences on new contents through effective interactive mechanisms (e.g., click, conversation).

Most existing solutions model the interactive recommendation problem as a contextual bandit problem to address the explore-exploit dilemma at a per-user basis. Li *et al.* [16] first modeled the contextual bandit problem and designed the LinUCB algorithm to learn an item-selection strategy based on user-click feedback to maximize long-term total user clicks. Wang *et al.* [26] and Zhou *et al.* [28] further considered learning the hidden features to leverage the performance of LinUCB algorithm. Unfortunately, these approaches require a large number of user interactions.

Recent researches focus on reducing the number of iterations by using collaborative learning across users. Gentile *et al.* [9] proposed a clustering bandit algorithm that adaptively clusters the bandit strategies. More works further improved the performance by taking into account the collaborative effects between users to dynamically grouping users and items simultaneously [18, 19, 21, 27]. Later on, Christakopoulou *et al.* [5] modeled the latent item space offline using collaborative filtering over all user data and then used them to initialize the model parameters on known items before interactions with user. However, these approaches heavily rely on the existence of large amount history data of each item from many users, which cannot be effectively applicable to dynamically changing items. Moreover, users oftentimes are reluctant to share their data due to their privacy concerns [22].

Our contributions: We design a novel *history state tracking* enabled contextual bandit algorithm to facilitate the policy learning for item selection in each round. Intuitively, the maintained history state models and remembers the features of selected items and feedback rewards from previous interactions. Thanks to the nature of exploration in contextual bandit problem, our approach can quickly learn a user's preferences on dynamically changing items by observing the user's feedback on a small number but large diversity of items. As a result, the incorporation of history state in our approach largely mitigates the required number of interactions with each user. To model the history state, we couple deep neural networks with the external neural memory, motivated by its capability to quick remember new information in its application of one-shot learning [20]. As such, our approach is referred to as *Deep neural Memory augmented Contextual Bandit* (DMCB).

In order to train DMCB model, we first design an offline training algorithm based on *a novel mini-batch construction mechanism* such that any user or item data can be used altogether as a large scale training set to train a high quality DMCB base model offline. For each single user, we further design an online training algorithm

RecSys '18, October 2–7, 2018, Vancouver, BC, Canada

ACM ISBN 978-1-4503-5901-6/18/10…$15.00
https://doi.org/10.1145/3240323.3240344

focusing on policy evaluation to fine tune and personalize DMCB. We devise novel *transition instances* to construct the replay buffer and fine tune the model using experience replay. A well evaluated policy is shown to be able to quickly improve when being paired with a simple decayed ϵ greedy algorithm. While deep learning based approaches focus on empirical results rather than theoretical bounds in general, we extensively validate our approach using both synthetic and large-scale real-world datasets.

The rest of paper will be organized as follows: Related work is discussed in Section 2. Section 3 presents our proposed DMCB model, the key idea and detailed model design, as well as both offline model training algorithm and online model fine-tuning for policy evaluation. The experimental results are shown in Section 5. Section 6 discusses the future work.

2 RELATED WORK

Contextual bandit problem has been widely used to model interactive recommendation. One class of approaches learn the policy based on the estimated reward of each action. Li *et al.* [16] developed the first LinUCB approach to select an action with highest upper confidence bound, with its theoretical analysis in [6]. Valko *et al.* [24] proposed KernelUCB that models the reward function with some kernel function. Allesiardo *et al.* [3] designed a neural network based contextual bandit algorithm. Recently, researches intended to use multiple users' data to improve the prediction performance. Gentile *et al.* [9] proposed a clustering bandit algorithm (CLUB) that adaptive clustering of bandit strategies. More works further considered the collaborative effects to dynamically grouping contexts and actions simultaneously [18, 19, 21, 27]. Zhou *et al.* [28] considered the latent correlation between contexts to learn the policy faster under new contexts. Agrawal *et al.* [2] proposed a Thompson sampling algorithm with linear rewards. [23] and Krause *et al.* [14] designed GP-UCB and CGP-UCB approaches that puts a Gaussian Process prior on the reward estimation function to encode the assumption of smoothness. Kawale *et al.* [13] developed Particle Thompson Sampling for MF (PTS) that selects an action based on Thompson Sampling in online matrix factorization. The other class of work focused on policy learning with exponentially or infinitely large set of policies. Many approaches were developed based on a natural abstraction from supervised learning for efficiently finding a function in a rich function class that minimizes the loss on a training set [7, 15]. Langford *et al.* [15] designed the epoch-greedy algorithm by uniformly exploring and then exploiting in each epoch. Dudik *et al.* [7] designed RandomizedUCB to assign a non-uniform distribution over policies to achieve a optimal regret. Agarwal *et al.* [1] further introduced ILOVETOCONBANDITS to reduce the time complexity of RandomizedUCB. Feraud *et al.* [8] built a random forest based on the known structure of policies. However, these approaches require many interactions with each user to provide high quality recommendations on dynamically changing items.

3 DEEP NEURAL MEMORY AUGMENTED CONTEXTUAL BANDIT (DMCB)

In this section, we first recall the contextual bandit problem. We then present our key idea in our approach and its underlying intuition. Afterwards, we discuss the model details in each component.

3.1 Contextual Bandit (CB)

Let $\mathcal{D}$ be an unknown probability distribution over the $X \times [0,1]^{r \times K}$, for a space of all contexts X and a finite set of K actions A. The rewards of actions are always in the interval $[0, 1]$. In each round t, the agent observes a context $\mathbf{x}_t$ and chooses an action $a_t \in A$ and then receives the r-dimensional reward $\mathbf{r}_{u,a_t} \in [0,1]^r$ of the selected action a_t. The goal is to minimize the regret $R(T)$ after T rounds:

$$R(T) = \sum_{t=1}^{T} E\|\mathbf{r}_{u,a_t^*} - \mathbf{r}_{u,a_t}\|_2^2 \quad (1)$$

where a_t^* is the optimal action in each round t. A policy $\pi : \mathbf{x}_t \rightarrow A$ is defined as the decision to select an action from all actions A under a context $\mathbf{x}_t$. The goal is to find the optimal policy π^* that minimizes regret $R(T)$. *Note that, unlike the standard supervised learning, the contextual bandit only sees the reward of selected actions.*

In the application of interactive recommendation, we follow the literature [16] to model a context $\mathbf{x}_t = \{\mathbf{x}_{u_t,a}\}_{\forall a \in A}$, in which $\mathbf{x}_{u_t,a}$ is the combination of user feature $\mathbf{x}_{u_t}$ of observed user u_t and item feature $\mathbf{x}_a$ of the item a in item pool A. That said, each user is corresponding to a context. Each item a is modeled as an action associated with a reward $\mathbf{r}_{u,a}$ from the feedback of user u_t. Our goal is to learn a policy that decides which item to select to recommend to a user in each round. In the rest of paper, two pairs of terms (user and context; action and item) will be used interchangeably.

3.2 Our Proposed DMCB Approach

Algorithm 1 DMCB Approach

1: ▷ offline learning cross users
2: Offline DMCB training using history data of all users;
3: ▷ online learning for a user u
4: **for** $t \leftarrow 1, \ldots, T$ **do**
5: The agent observes the context $\mathbf{x}_t = \{\mathbf{x}_{u,a}\}_{\forall a \in A}$ for the user u;
6: The agent chooses an item $a_t = \pi(\mathbf{x}_t)$ based on the history state S_{t-1}^u;
7: The reward $\mathbf{r}_{u,a_t}$ of select item a_t is revealed;
8: The agent updates the history state S_t^u;
9: Online DMCB training for evaluating policy π_u based on the history state S_{t-1}^u and the reward $\mathbf{r}_{u,a_t}$ of selected action a_t;
10: **end for**

Algorithm 1 shows the overview of our proposed DMCB approach, in which the brown line shows the novel component in DMCB model and the blue lines correspond to our proposed training algorithms for DMCB model. As one can see, in each round, the key difference from our DMCB model from existing approaches is that it maintains the history state S_t and uses it to decide which item to select in next round. *As we argued in introduction, the history state can quickly learn to understand a user's preferences by remembering the features of diversely explored items in few interactions with this user. As such, our proposed DMCB model can largely overcome the major challenge in contextual bandit problem that it can only observe the reward of selected actions.* In order to train DMCB model, we first propose an offline DMCB training algorithm which utilizes the large amount of history data from all users as training set. Then, we design an online training algorithm for on-the-fly model finetuning for each user to personalize the model. We focus on policy evaluation using experience replay and pair it with a simple decayed ϵ greedy bandit strategy to select action in the next round. Note that our online training can be simply applied to interleaved users by maintaining different states for each user.

Remarks: The key difference between contextual bandits and reinforcement learning is as follows: the contexts in contextual bandits do not depend on the past actions; while the contexts in reinforcement learning do depend on the past actions through the modeled states via observed environment. In this paper, our DMCB model is designed to use external neural memory to model and track a latent history state based on previous actions in contextual bandits. Thus, we augment the contexts implicitly using latent states although there is no explicitly observable environment. As a result, DMCB can use such dynamic history state to quickly learn the policy with a small number of interactions.

3.3 History State Design using Neural Memory

As the key of DMCB model design, we model *history states* motivated by the recent success of Neural Turing Machine (NTM) [11] in various applications such as one-shot learning [20], copying, sorting data sequences and so on. As an end-to-end fully differentiable architecture designed to extend the capabilities of neural networks by coupling them with external memory, NTM has demonstrated its capability to quickly capture new information based on the memorized state in one-shot learning. Therefore, the application of NTM idea can intuitively accelerate the policy learning for contextual bandits with reduced number of interactions.

In DMCB, we model the history states S_t at round t (simplied notation of S_t^u for user u) with two components: (1) *external memory state* S_t^m: consists of selected item cells $\mathbf{MI}_t$ and observed reward cells $\mathbf{MR}_t$ to store the latent features of the selected items and the corresponding observed rewards in previous interactions; (2) *controller state* S_t^c: represents the state of central controller to guide the interaction with external memory in DMCB model.

Our proposed DMCB model with novel history states enjoys two advantages: (1) Thanks to the exploration nature in CB problem, DMCB can quickly and effectively understand each user's personal preferences by selecting and observing a smaller number but large diversity actions and feedback rewards during the first few iterations. (2) DMCB makes it possible to share the model between different users and still maintain each user's personal preference information in the latent history states in external memory.

Remarks: It is important to mention that NTM can be used to treat the data either as a sequence or as a set [25]. Thus, it can be suitably applied onto contextual bandits.

3.4 DMCB Model Architecture

Figure 1 shows the architecture of our proposed DMCB model. In each round t, DMCB takes the context $\mathbf{x}_t$ to estimated value $\mathbf{v}_{t,a}$ for each item $a \in A$ based on the current history state $S_{t-1} = (S_{t-1}^m, S_{t-1}^c)$; then selects an item a_t and observes its reward $\mathbf{r}_{u,a_t}$; at last uses a_t and $\mathbf{r}_{u,a_t}$ to update the history state from S_{t-1} to S_t.

The main difference in our DMCB model from NTM [11] and modified NTM [20] is two fold: (1) We split the external memory into item and reward cells and treat them differently; (2) We introduce a modified LRUA to ensure all empty memory cells are filled before overwriting at the beginning rounds for each user. Next, we introduce each module in details.

External Memory: consists of *item* and *reward* cells to store the latent features and the received rewards of select items. The item cell is used to address the locations based on the observed context. At round t, external memory state S_t^m has an item state $\mathbf{MI}_t$ and a reward state $\mathbf{MR}_t$ (Figure 1).

Controller: guides the read head and write head to access external memory and estimates values. We model it as *Long Short-Term Memory* (LSTM) [10] since its internal memory can avoid the long-term dependency problem in recurrent neural networks (RNN). When the controller observes the context $\mathbf{x}_t$, it invokes the read head to extract history state in external memory; and uses it to estimate the values. When the controller observes the selected item $\mathbf{x}_{u,a_t}$ and its reward $\mathbf{r}_{u,a_t}$, it invokes the write head to update the history state of external memory S_t^m and the controller itself S_t^c. In each round t, controller state S_t^c is composed of a hidden state $\mathbf{ch}_t$ and a cell state $\mathbf{cc}_t$ which behaves as a conveyor belt over time in LSTM. At each round, the LSTM controller update its states $\mathbf{cc}_t, \mathbf{ch}_t$ as follows:

$$\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} sigmoid \\ sigmoid \\ sigmoid \\ tanh \end{pmatrix} \begin{pmatrix} \mathbf{W}_i^l \\ \mathbf{W}_f^l \\ \mathbf{W}_o^l \\ \mathbf{W}_c^l \end{pmatrix} \begin{pmatrix} \mathbf{Z} \\ \mathbf{ch}_{t-1}^l \end{pmatrix} + \mathbf{b}_h$$

$$\mathbf{cc}_t^l = \mathbf{f} \odot \mathbf{cc}_{t-1}^l + \mathbf{i} \odot \mathbf{g}$$

$$\mathbf{ch}_t^l = \mathbf{o} \odot tanh(\mathbf{cc}_t^l)$$

where $\mathbf{cc}_{t-1}, \mathbf{ch}_{t-1}$ are obtained from saved controller states at previous iteration $t-1$; $\mathbf{i}, \mathbf{f}, \mathbf{o}$ are the input gates, forget gates, output gates respectively; $\mathbf{W}_i^l, \mathbf{W}_f^l, \mathbf{W}_o^l, \mathbf{W}_c^l$ are the weights of layer l in LSTM; and $\odot$ represents element-wise multiplication. In terms of the input $\mathbf{Z}$, we choose $\mathbf{Z} = \mathbf{h}_t^{l-1}$ for any stacked deep layer of LSTM. For the first layer, $\mathbf{Z} = \mathbf{x}_{u,a}$ for the item a with interitem with user u at round t.

Read Head: Given the output $\mathbf{ch}_t$ from LSTM controller, the goal of read head is to address the external memory and fetch the item and reward memory states ($\mathbf{ri}_t$ and $\mathbf{rr}_t$) respectively. Then, it uses them to estimate the value $\mathbf{v}_{u,t,a}$ of item a in the observed context $\mathbf{x}_{u,a}$.

As described above, the memory addressing will only happen in the item memory cell in external memory. In our model, we consider using the content based memory addressing proposed in [20]. First, DMCB generates a key query vector w.r.t. the context memory from the output $\mathbf{ch}_t$ of LSTM controller:

$$\mathbf{k}_t^c = tanh(\mathbf{W}_k^c \mathbf{ch}_t)$$

We compute the cosine distance between $\mathbf{k}_t^c$ and each row of item cell as follows:

$$K(\mathbf{k}_t^c, \mathbf{MI}_t(i)) = \frac{\mathbf{k}_t^c \cdot \mathbf{MI}_t(i)}{\|\mathbf{k}_t^c\| \|\mathbf{MI}_t(i)\|}$$

A read wright vector $\mathbf{w}_t$ is then calculated using the softmax over $K(\mathbf{k}_t^c, \mathbf{MI}_t(i))$:

$$\mathbf{w}_t^r(i) = \frac{\beta_t \exp K(\mathbf{k}_t^c, \mathbf{MI}_t(i))}{\sum_j \beta_t \exp K(\mathbf{k}_t^c, \mathbf{MI}_t(j))}$$

where β_t is a positive key strength which can amplify or attenuate the precision of the focus:

$$\beta_t = \log(\exp(\mathbf{W}_\beta \mathbf{ch}_t) + 1)$$

such that $\beta_t \geq 1$ for any t.

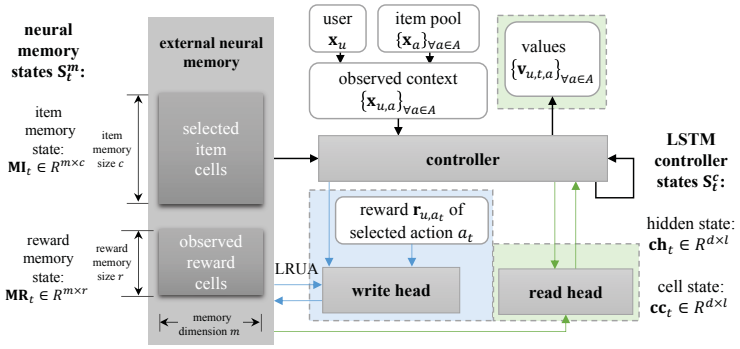

Figure 1: DMCB Model Architecture (Round t with User u): (1) The external memory consists of item and reward cells, which store the latent factors of selected items and observed feedback rewards respectively; (2) The controller, behaving as a *"brain"* of DMCB, instructs read and write heads as well as estimates the values; (3) The green component takes the observed context $\{x_{u,a}\}_{\forall a \in A}$ as input; then uses the read head to estimate the values $\{v_{u,t,a}\}$ of each item a; (4) The blue component takes the pair of selected item $x_{u,a}$ and its reward $r_{u,a}$ as input; then uses the write head via blue arrows to update the history state of controller and external memory.

After getting the controlling weight vector of read head $\mathbf{cw}_t^{\mathbf{r}}$, we retrieve $\mathbf{ri}_t$ and $\mathbf{rr}_t$ as follows:

$$\mathbf{ri}_t = \mathbf{w}_t^{r\,T}\mathbf{MI}_t, \quad \mathbf{rr}_t = \mathbf{w}_t^{r\,T}\mathbf{MR}_t$$

When there are more than one read head, $\mathbf{k}_t^c(i)$ will be learned for each read head i. And $\mathbf{ri}_t, \mathbf{rr}_t$ will be set as $\sum_i \mathbf{ri}_t(i)$ and $\sum_i \mathbf{rr}_t(i)$ respectively.

Write Head: takes charge of the update of the memory. At the beginning of interitem with each user, the memory is empty (initialized as all **0**). We modify *Least Recently Used Access* (LRUA) in [20] to ensure that the memory will not be overwritten until no memory rows are empty. LRUA [20] addresses the write locations by balancing the most relevant read location or the least recently used location, such that the memory can keep up-to-date and diverse. At the first round, the most recently read location weight $\mathbf{w}_t^u$ is set as:

$$\mathbf{w}_{t=0}^u = (0, 0, \cdots, 0)$$

where the size of $\mathbf{w}_{t=0}^u$ is the number of rows in memory m. Then, the least-used weights, $\mathbf{w}_{t-1}^{lu}$ is computed as the same as in [20]:

$$\mathbf{w}_{t-1}^{lu}(i) = \begin{cases} 0 & \text{if } \mathbf{w}_{t-1}^u(i) > m(\mathbf{w}_{t-1}^u, n) \\ 1 & \text{if } \mathbf{w}_{t-1}^u(i) \le m(\mathbf{w}_{t-1}^u, n) \end{cases}$$

where $m(\mathbf{w}_{t-1}^u, n)$ denote the n^{th} smallest element of the vector $\mathbf{w}_{t-1}^u$ and n is the number of read heads.

In order to ensure that the entire memory is fulfilled before overwriting the non-empty rows, we introduce another last read weight vector $\mathbf{w}_{t-1}^{rr}$ as follows:

$$\mathbf{w}_{t-1}^{rr}(i) = \begin{cases} 0 & \text{if } \mathbf{w}_{t-1}^u(i) \le m(\mathbf{w}_{t-1}^u, n) \text{ and } \mathbf{w}_{t-1}^u(i) = 0 \\ \mathbf{w}_{t-1}^u(i) & \text{otherwise} \end{cases}$$

where $\mathbf{w}_{t=0}^r = \mathbf{0}$ initially.

Then, the write weights $\mathbf{w}_t^w$ is similarly defined as a convex combination between with a trainable sigmoid gate as follows:

$$\mathbf{w}_t^w = sigmoid(\alpha)\mathbf{w}_{t-1}^{rr} + (1 - sigmoid(\alpha))\mathbf{w}_{t-1}^{lu}$$

At last, we update the location weight vector $\mathbf{w}_t^u$:

$$\mathbf{w}_t^u = \mathbf{w}_{t-1}^u + \mathbf{w}_t^w$$

Accordingly, we update the item and reward memory states by erasing and writing into the locations via $\mathbf{w}_t^w$. In erasing phase, the memory is updated as follows:

$$\mathbf{MI}_t = \mathbf{MI}_{t-1} \odot [(\mathbf{1} - \mathbf{w}_t^w) \otimes \mathbf{1}]$$
$$\mathbf{MR}_t = \mathbf{MR}_{t-1} \odot [(\mathbf{1} - \mathbf{w}_t^w) \otimes \mathbf{1}]$$

That said, the memory in located addresses only remains the leftover portion with respect to the write weight $\mathbf{w}_t^w$. In writing phase, the memory is updated as follows:

$$\mathbf{MI}_t = \mathbf{MI}_{t-1} + \mathbf{w}_t^w \otimes \mathbf{k}_t^c$$
$$\mathbf{MR}_t = \mathbf{MR}_{t-1} + \mathbf{w}_t^w \otimes \mathbf{k}_t^r$$

for each cell i in external memory. Since the reward cells in external memory are used for estimating the value rather than memory addressing, we choose $\mathbf{k}_t^r$ to be the observed reward $\mathbf{r}_{u,a_t}$ of the selected item a_t.

Value Esimation Output: aims to estimate the value of each item a given the observed context $\mathbf{x}_{u,a}$. Specifically, the estimated reward is calculated as a linear projection from the output of reward memory from the read head:

$$\mathbf{v}_{u,t,a} = \mathbf{W}_o\mathbf{rr}_t$$

4 DMCB TRAINING ALGORITHMS

4.1 Offline DMCB Training

We first design an offline DMCB training algorithm that enables to utilize the large amount of history data from all users' history data. As DMCB offline training is only based on the previously selected items and observed rewards, it does not involve policy learning on how to select items in each round.

Fortunately, thanks to the external memory that can store different data for each user, this offline training of DMCB model is mainly to train the read and write capabilities of the model such that it can

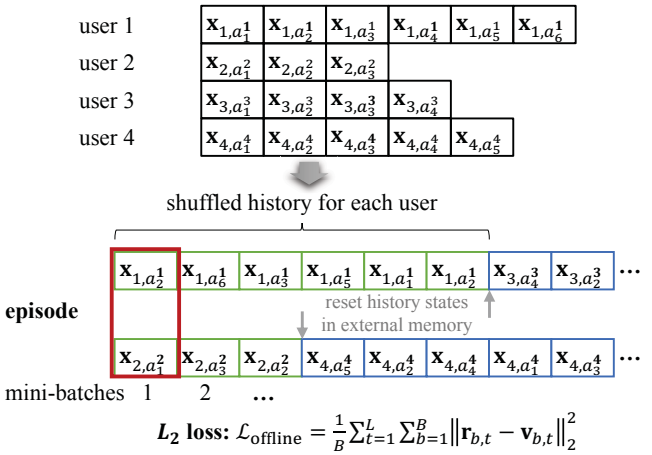

Figure 2: Mini-Batch Construction across Users for DMCB Offline Training: $\mathbf{r}_{b,t}, \mathbf{v}_{b,t}$ are the observed reward and estimated value of the t^{th} item in the b^{th} sample of a mini-batch in the episode.

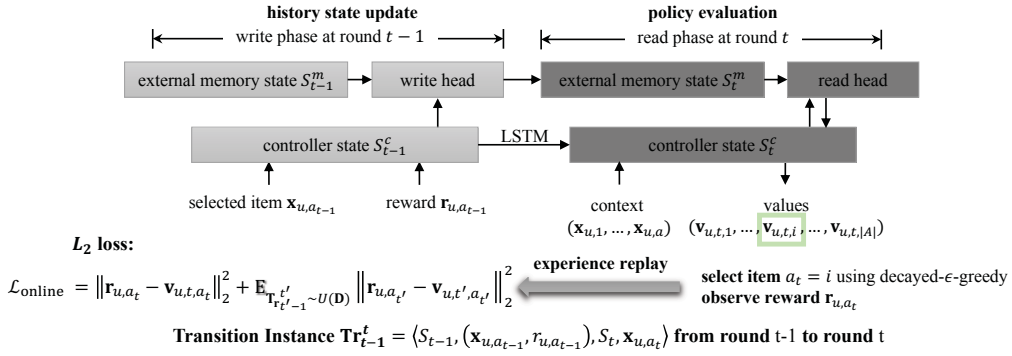

Figure 3: Transition Instance Construction for DMCB Online Training: We consider each transition instance starting with the write phase at round $t-1$ and ending with the read phrase at round t. When the agent observes the reward $\mathbf{r}_{u,a_{t-1}}$ of the selected item a_{t-1} at round $t-1$, the controller invokes write head that updates the states of external memory from S^m_{t-1} to S^m_t. In the meanwhile, the LSTM controller updates its own state from S^c_{t-1} to S^c_t. With the new states of external memory and LSTM controller, the new context $\{\mathbf{x}_{u,a}\}_{a\in A}$ are observed to start with round t. Then, the controller invokes read head to estimate values of all items and uses the current policy to select an item a_t (in green). At last, the observed reward $\mathbf{r}_{u,a_t}$ will be the label of this transition instance $\mathbf{Tr}^t_{t-1}$.

later be used to efficiently store previous interitems with a user and quickly learn this user's preferences.

However, the construction of mini-batches is still non-trivial since the external memory is not trainable during the parameter updating of DMCB model. Thus, it is critical to design the mini-batches, rather than randomly sample the pairs of item and reward, such that the states in external memory can be properly tracked during DMCB training to help estimate the values in next round. Moreover, due to the different number of items for each user, simple parallelization for different user's data cannot be directly used here. Therefore, we introduce an *episode* with selected items for one or more users as a sample within one mini-batch. Each episode has a fixed length L and the model parameters will be updated at the end of each episode.

Figure 2 shows our mini-batch construction approach. The idea is to concatenate the history for each user to form each episode of a mini-batch by reinitializing history states for every new user. At the beginning of each epoch, we first randomly select B (size of mini-batch) users among all unselected users. In each episode, after reshuffling and adding the selected items for one user, items for a new user will be concatenated afterwards with reinitialized history states in external memory cells and controller states. In each iteration of training, we estimate the value of selected item based on the previous history state. At the end of each episode, we update the model parameters using the L_2 loss in Figure 2. Each epoch finishes when all training data are selected. The whole offline learning runs K epochs in total. All users as well as the selected items in each user's history are then reshuffled to start with a new epoch. Algorithm 2 shows the details of offline cross-users DMCB model training.

Algorithm 2 Cross-Users DMCB Offline Training

```
for epoch = 1 . . . K do
    for episode = 1 . . . do                          ▷ Construct a mini-batch
        for sample b = 1, . . . , B do                ▷ Construct each sample
            if exist unfinished users from last episode then
                Randomly select an unfinished user u;
                Initialize external memory and controller states for these users
using the previously stored states;
            else
                Randomly select a new user u from all unselected users;
                Initialize external memory and controller state for these users to 0;
            end if
            Add all items for user u into sample b;
            ▷ Concatenate a new user to sample b
            if the number of items are less than L then
                Reinitialize external memory and controller states to 0;
                Randomly select a new user in training data;
            end if
        end for
        Run RMSProp Optimizer to minimize L_offline to update DMCB parameters;
        Store the ending states of all users in this mini-batch;
    end for
end for
return DMCB model with parameters θ
```

4.2 Per-User Online DMCB Training for Policy Evaluation & Policy Learning

Since the history states in external memory is independent of DMCB training, it also becomes harder to better evaluate and quickly improve the policy with continuously updated states. In this paper, we focus on designing a good policy evaluation mechanism via online DMCB fine-tuning and pair it with a simple decayed ϵ-greedy bandit strategy. That said, we select an item a_t at round t with highest value $\mathbf{v}_{u,t,a_t}$ derived from evaluated policy ($a_t = \arg\max_{a\in A} \mathbf{v}_{u,t,a}$) with probability $1-\epsilon$ and randomly select an item otherwise. ϵ is decayed after each round. While ϵ-greedy algorithm might be considered relatively inefficient in literature, we show later in experiments its effectiveness when incorporated with our policy evaluation mechanism as well as memorized history states in DMCB model.

As the key component of our policy evaluation algorithm, the online DMCB fine tuning is conducted as follows: In each round t, DMCB first takes the pair of selected item and its observed reward $(\mathbf{x}_{u,a_{t-1}}, \mathbf{r}_{u,a_{t-1}})$ in previous round as input to update the memory and controller states using the write head. With the new states, the read head will be invoked to estimate the value $\mathbf{v}_{u,t,a}$ of each item

Algorithm 3 Online DMCB Training for User u

1: Initialize external memory state S_1^m; ▷ Initialize states
2: Initialize controller state S_1^c;
3: Load all parameters $\boldsymbol{\theta}$ from pre-trained DMCB model offline; ▷ Initialize model parameters
4: Initialize transition state set $D = \emptyset$;
5: ▷ Start iterative recommendation and online model training
6: Observe context $\{\mathbf{x}_{u,a}\}_{\forall a \in A}$ of all items $a \in A$; ▷ Run round $t = 1$
7: Randomly select an item a_1;
8: Observe reward $\mathbf{r}_{u,a_1}$;
9: **for** $t = 2 \ldots$ **do**
10: ▷ Update memory state
11: Write $(\mathbf{x}_{u,a_{t-1}}, \mathbf{r}_{u,a_{t-1}})$ into external memory via write head;
12: Save new external memory state S_t^m;
13: **for each** $a \in A$ **do** ▷ Update value of each item
14: Calculate $\mathbf{v}_{u,t,a}$ using current parameters $\boldsymbol{\theta}$ via read head;
15: **end for**
16: ▷ Select an item using ϵ-greedy algorithm
17: Select $a_t = \arg\max_{a \in A} \mathbf{v}_{u,t,a}$ with probability $1 - \epsilon$;
18: Randomly select an item $a_t \in A$ otherwise;
19: Observe reward $\mathbf{r}_{u,a_t}$; ▷ Observe reward
20: Decay ϵ by γ; ▷ Decay ϵ for next round
21: ▷ Store transition instance
22: Add transition instance Tr_{t-1}^t $(\langle S_{t-1}, (\mathbf{x}_{u,a_{t-1}}, \mathbf{r}_{u,a_{t-1}}), S_t, \mathbf{x}_{u,a_t} \rangle)$ to D
23: ▷ Per-Round Training
24: Randomly sample mini-batch B_t of transition instances from D;
25: Run RMSProp Optimizer to minimize loss $\mathcal{L}_{online}$ to update DMCB model parameters $\boldsymbol{\theta}$;
26: **end for**

a. Based on the estimated rewards $\{\mathbf{v}_{u,t,a}\}_{\forall a \in A}$, the decayed-$\epsilon$-greedy algorithm will be used to select an item a_t and the reward $\mathbf{r}_{u,a_t}$ of a_t is observed. At the end of each round t, a *transition instance* Tr_{t-1}^t between each round $t-1$ and t is constructed as shown in Figure 3: $\mathsf{Tr}_{t-1}^t = \langle S_{t-1}, (\mathbf{x}_{u,a_{t-1}}, \mathbf{r}_{u,a_{t-1}}), S_t, \mathbf{x}_{u,a_t} \rangle$ and $\mathbf{r}_{u,a_t}$ is assigned to the label of Tr_{t-1}^t. Tr_{t-1}^t will be stored in replay buffer D and will be used with sampled transition instances in D for experience replay to fine tune DMCB model by minimizing L_2 loss in Figure 3. The decay factor of exploration can be adjusted based on the preferred number of items for each user.

Algorithm 3 describes the details of online fine tuning for a user u. Line 1-2 sets both memory and controller states to be all zeros at the beginning of interitem with a new user. In line 3-5, DMCB is initialized using the parameters in above pre-trained offline model. In the first round, it randomly selects the first item and receives its reward. Specifically, in each round, the agent first writes the tuple of selected item and observed reward in the previous round into external memory; then estimates the rewards of all items and selects a new item using ϵ-greedy. At the end of each item selection, DMCB model is updated via experience replay, by using the current transition along with $B-1$ randomly sampled transitions from previous rounds. The objective loss function $\mathcal{L}_{online}$ in Figure 3 is minimized by using RMSProp optimizer. During the whole process, the agent continuously selects items and observes rewards of the selected items. More and more latent features and rewards of selected items can be quickly learned via the data stored in external memory to understand this user's preferences.

5 EXPERIMENTAL EVALUATION

5.1 Settings

Implementation Details: We implement DMCB model and training algorithms in TensorFlow using the following hyperparameters. The sizes of item and reward memory cells are 10×5 and 1×5. The controller has one layer LSTM with 10 hidden state dimension, along with one read head and one write head. In offline DMCB training, we choose mini-batch size as 4, with each episode of length 25. The offline training takes 10 epochs using learning with 0.005 with decay factor 0.95. In online DMCB training, we choose mini-batch size as 4 using learning with 0.001 with decay factor 0.95. The exploration parameter ϵ is evenly distributed with initial value 1 to final value 0.1 over all iterations, i.e., $\gamma = 0.03$ when there are 30 iterations. We use RMSPropOptimizer with gradient momentum 0.9.

Competitors: We compare with the following state-of-the-art bandit approaches:

- *LinUCB* [16]: selects an item based on an upper confidence bound of the estimated reward with linear relationship with user vectors;
- *Hybrid-LinUCB* [16]: extended LinUCB with some parameters shared between users;
- *Hidden-LinUCB* [26]: extended LinUCB with both observed and hidden features of each item;
- *PTS* [13]:selects an item based on Thompson Sampling in online matrix factorization;
- *UCB-PMF* [19]: selects an item based on a UCB-like strategy using matrix factorization;
- *CLUB* [9]: selects an item based on a single model for each group by adaptively clustering bandit policies;
- *CoLin* [27]: selects an item based on the joint learned bandit that models latent dependency among users.

The latent feature dimension in hLinUCB is set to 5, and in UCB-PMF and PTS to 10. Since they do not have states of different users to track, we run them separately from scratch under each user.

5.2 Evaluation on Synthetic Datasets

Synthetic Datasets: We design two types of simulators to generate synthetic datasets and each simulator runs 30 rounds under each user with 1000 users in total. One type is *Linearly-Rewarded Simulator* with linearly correlated items and rewards; the other is *Collaborative Simulator* in which the rewards are determined collaboratively by all users.

Before introducing the details of each simulator, we first generate an shared 1000 users and an item/action pool A of size $K = 30$. Each item $a \in A$ is associated with d-dimensional feature vector $\mathbf{x}_a$ with $d = 25$. Each d-dimensional item feature vector is drawn uniformly at random on the surface of a Euclidean unit ball. That said, we sample each point in $z_1, \ldots, z_d$ independently from Gaussian distribution $N(0,1)$ and normalize each point by $\sqrt{z_1^2 + \cdots + z_d^2}$. In the same way, we model each user with a 5-dimensional feature vector $\mathbf{x}_u$. At the end, $\mathbf{x}_{u,a}$ is set to be the concatenation of $\mathbf{x}_u$ and $\mathbf{x}_a$, i.e., $\mathbf{x}_{u,a} = \mathbf{x}_u \oplus \mathbf{x}_a$.

(1) Linearly-Rewarded Simulator: we consider two simulators, *LinUCB Simulator* in [16] and *Latent LinUCB Simulator* in [26]. In both simulators, we model optimal bandit for each user u using a d-dimensional vector $\boldsymbol{\theta}_u^*$. Each dimension of $\boldsymbol{\theta}_u^*$ is drawn from a uniform distribution $U(0,1)$ and normalized to $\|\boldsymbol{\theta}_u^*\| = 1$. The ground-truth reward $\mathbf{r}_{u,a}$ is $\mathbf{x}_a^T \boldsymbol{\theta}_u^* + \eta$ where η is drawn from a Gaussian noise $N(0, \delta^2)$ with $\delta = 0.1$. In each round t, LinUCB

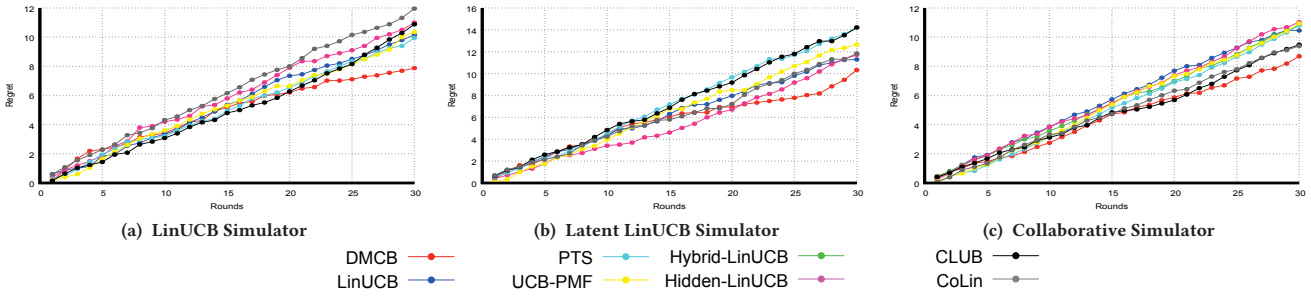

Figure 4: Results on Synthetic Datasets

simulator and Latent LinUCB simulator observe the full context ($\{\mathbf{x}_{t,a}\}_{a\in A}$) and the partial context (the first 20 dimensions of each $\mathbf{x}_a$) respectively; and both receive the same reward $\mathbf{r}_{u,a_t}$ of the selected item a_t.

(2) Collaborative Simulator: we follow [27] to model optimal bandit as follows. We define the golden relational stochastic matrix $\mathbf{W}$ for the graph of users is then constructed with $w_{ij} \propto \langle \boldsymbol{\theta}_i^*, \boldsymbol{\theta}_j^* \rangle$ in which each column is normalized by its L_1 norm. The ground-truth reward $\mathbf{r}_{u,a}$ is $diag(\mathbf{x}_a^T \boldsymbol{\theta}_u^* \mathbf{W}) + \eta$ where η is drawn from a Gaussian noise $N(0, \delta^2)$ with $\delta = 0.1$. user features for Hybrid-LinUCB algorithm is constructed by using the first 5 principle components on the matrix $\mathbf{W}$ via PCA. In each round t, the agent observes the user $\{\mathbf{x}_{t,a}\}_{a\in A}$ and receives the reward $\mathbf{r}_{t,a_t}$ of the selected item a_t.

Evaluation Results: Among 1,000 generated users, we randomly select 900 users for DMCB offline training. Then using these DMCB models, we test our online algorithm on the rest of 100 users, considered as new users, with 30 rounds for each user. We skip the report of offline training results that is similar as in real-world datasets in the next subsection. Figure 4 shows the regret averaged on all testing users. As one can see, our proposed DMCB approach consistently performs better than competitors. This is because the track of history states in DMCB largely leverages the observed user to quickly learn the policy. DMCB outperforms competitors up to around 20% after 30 rounds in LinUCB Simulator than other simulators since the fully observable users by themselves provide more information to the agent. The performance advantage of DMCB over competitors in collaborative simulator is smallest as the correlation between items helps to accelerate the policy learning.

5.3 Evaluation on Real-World Datasets

Datasets: We evaluate on three benchmark datasets.

(1) Yahoo Today Module Dataset: user interitem history with Yahoo Today Module in a ten-day period in May 2009, which contains 45,811,883 user visits. For each visit, both the user (context) and each of the 10 candidate articles (items) are associated with a feature vector of six dimensions (including a constant bias feature). As there is no user identity in this data set, we first generate the user identities by assuming each user has a unique feature vector using the same mechanism in [26, 27]. Then we generate the userual features as follows: we first apply K-means algorithm to cluster all users into user groups based on the given user features and assign each record to its closest user group. For each user, we construct the context as a set of feature vectors in which each feature vector corresponds to an item. Specifically, for each item, the feature vector is the concatenating the cluster centroid vector of this user and the article feature vector of this item.

(2) Last.FM Dataset: a music artist listening data from Last.FM online interactive music recommender system. Each artist is tagged by multiple users with totally 11,946 tags. It contains 1,892 users (users) and 17,632 artists (items), as well as 92,834 user-listened artist count pairs.

(3) Delicious Dataset: a bookmarking dataset from Delicious interactive social system with social networking, bookmarking, and tagging information from a set of users. Each bookmark is tagged by multiple users with totally 53,388 tags. It contains 1,867 users (users), 69,226 bookmarked URLs (items), 104,799 user bookmarking relations.

Following the previous work [4], we construct the reward and user features in Last.FM and Delicious datasets as follows: For each user, the reward is 1 if the user listened to an artist or bookmarked a particular URL, otherwise 0. The features of all artists/bookmarked URLs are calculated using TF-IDF (term frequency–inverse document frequency) algorithm on the assigned tags. User features were not available because of privacy issues. Thus, the observed user features are the same under each user, i.e., $\mathbf{x}_{t,a} = \mathbf{x}_a$ for any t. In both datasets, we only retained the first 25 principle components to construct the user feature vectors as in [26, 27]. This also helps reduce the overfitting when running competitors with a small number of rounds.

Evaluation Results: we report the DMCB offline training and online policy evaluation respectively.

Offline DMCB Training: In each dataset, we randomly select 512 users for offline training. Then we report the L_2 loss on each mini-batch at the beginning of each episode. We report the averaged loss of every 10 episodes with totally 500 episodes. Figure 5a shows the result. As one can see, the loss (defined in Figure 2) decreases to less than 1 at the end of all episodes. That said, the averaged loss of each observed reward is less than 0.1. The loss fluctuates over time because of the various number of items under different users. The state reset at the beginning of each user will lead to the increased losses in the following several rounds until sufficient information is stored in the state to better estimate the values of items. Among all these datasets, Delicious dataset fluctuates most and converges slower due to the data sparsity [4] and frequently reset of states.

Online DMCB Training and Policy Evaluation: Using the offline trained DMCB models for each dataset, we test our online algorithm on the rest of users. We treat these testing users as new users that

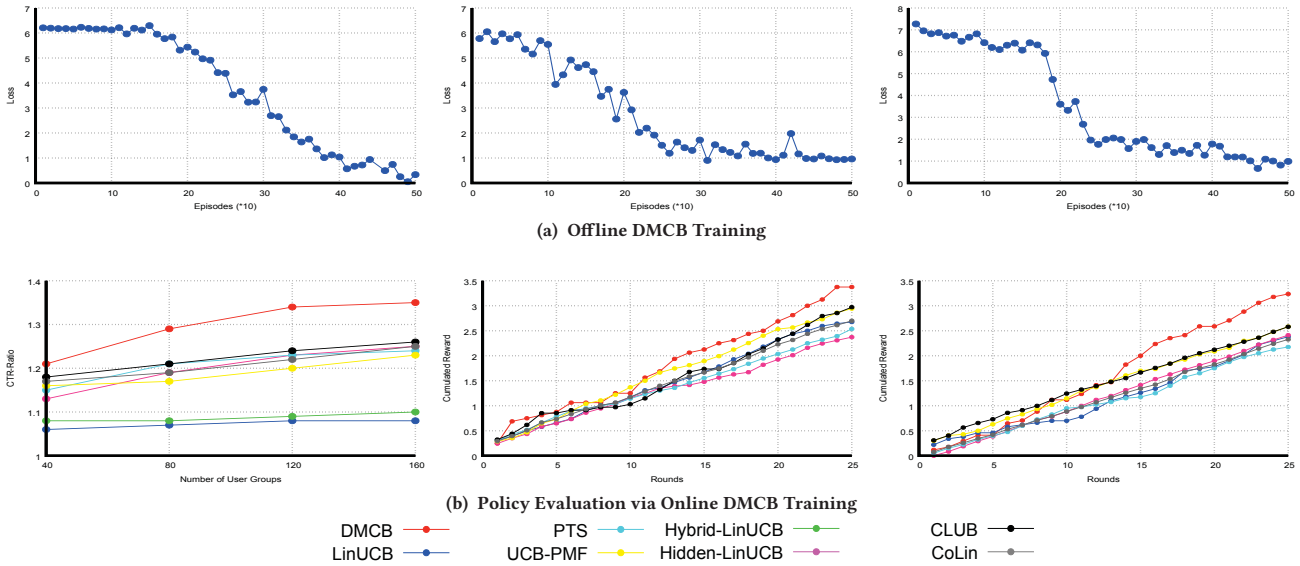

Figure 5: Results on Real-World Datasets (Left: Yahoo Dataset, Middle: Last.FM Dataset, Right: Delicious Dataset)

we do not have any information beforehand. We study the first 25 iterations per user according to the real world statistics [12].

In Yahoo dataset, we use the unbiased offline evaluation in [17] to evaluate the performance of different approaches. We measure Click-through-rate (CTR) as our metric, which is defined as the ratio between the number of clicks the agent receives and the number of recommendations it provides. Following the same evaluation strategy in [16], we use relative CTR normalized by the corresponding logged random strategy's CTR. We use CTR standing for relative CTR for simplicity. Figure 5b (left) reports CTR with different number of user groups after 25 rounds of user iterations. We study the impact of different granularities of observed users to the performance of algorithms. As one can see, our DMCB approach consistently outperforms existing approaches by 5% to 8%. With larger number of user groups (more user information observed), the advantage is more significant due to the capability of history state in DMCB to store these information.

In Last.FM and Delicious datasets, for each user, we use the same setting in [26, 27] to fix the size of candidate item pool. Since we test on small iterations, we fix item pool size to be 10 by picking one item from those non-zero-reward items according to the whole observations in the dataset and randomly picking the other 9 from those zero-reward items. The middle and right figures in Figure 5b show the results of cumulated rewards in Last.FM and Delicious datasets respectively. We can see that our DMCB starts to significantly outperform existing approaches after 12 and 14 iterations respectively. This is because the history states in DMCB after these rounds have largely augmented the user to accelerate the policy learning, however the competitors can learn policy only from observed users. It is interesting but not surprising that our results are consistent with one-shot learning in [20] where the model only needs to see very few samples in a new class to correctly classify other samples in this class.

At last, we report the diversity of selected items using different approaches. Our DMCB is shown to better exploit known information instead of slowly and continuously exploring new information than the competitors after around 15 iterations. Among all selected items between round 15 and 25, 55%-65% items are new in DMCB while still at least over 75% are new in competitors. More importantly, thanks to the ability of DMCB to quickly remember these explored items, it is flexible to select ϵ and its decay factor in decayed-ϵ-greedy algorithm based on the user preferred number of iterations in different practical applications, rendering DMCB much applicable in interactive recommendations.

6 DISCUSSION & FUTURE WORK

In this paper, we studied interactive recommendation in an online setting by modeling it as contextual bandits problem. We designed DMCB with deep neural history states to track each user's history such that this user's preferences on dynamic changing items can be quickly learned only with a small number of interactions. We introduced a novel cross-user offline training to bootstrap the DMCB model. Then, we focused on policy evaluation by tracking history states in DMCB and pair it with a standard bandit strategy. We showed the significant performance improvement in the setting of discretized actions. In addition to adapting dynamic changing items, we also showed in the experiment that our approach can also tackle completely new users. While our general setting does not require discretized actions and our proposed approach become less effective with large action space, our immediate next step is to improve DMCB to design an end-to-end bandit policy learning with latent policy evaluation. In this way, our method can be scaled up to support large action space. Moreover, as the NTM can also support sequential data, we will extend DMCB model into sequence modeling and policy evaluation for Markov decision processes in reinforcement learning.

REFERENCES

[1] Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert E. Schapire. 2014. Taming the Monster: A Fast and Simple Algorithm for Contextual Bandits *(ICML'14)*. II–1638–II–1646.

[2] Shipra Agrawal and Navin Goyal. 2013. Thompson Sampling for Contextual Bandits with Linear Payoffs *(ICML'13)*. 127–135.

[3] Robin Allesiardo, Raphaël Féraud, and Djallel Bouneffouf. 2014. A Neural Networks Committee for the Contextual Bandit Problem *(NIPS'14)*. 374–381.

[4] Nicolò Cesa-Bianchi, Claudio Gentile, and Giovanni Zappella. 2013. A Gang of Bandits *(NIPS'13)*. 737–745.

[5] Konstantina Christakopoulou, Filip Radlinski, and Katja Hofmann. 2016. Towards Conversational Recommender Systems *(KDD'16)*. 815–824.

[6] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. 2011. Contextual Bandits with Linear Payoff Functions *(AISTATS'11)*. 208–214.

[7] Miroslav Dudik, Daniel Hsu, Satyen Kale, Nikos Karampatziakis, John Langford, Lev Reyzin, and Tong Zhang. 2011. Efficient Optimal Learning for Contextual Bandits *(UAI'11)*. 169–178.

[8] Raphaël Féraud, Robin Allesiardo, Tanguy Urvoy, and Fabrice Clérot. 2016. Random Forest for the Contextual Bandit Problem *(AISTATS'16)*. 93–101.

[9] Claudio Gentile, Shuai Li, and Giovanni Zappella. 2014. Online Clustering of Bandits *(ICML'14)*. 757–765.

[10] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. 2000. Learning to Forget: Continual Prediction with LSTM. *Neural Comput.* 12, 10 (Oct. 2000), 2451–2471.

[11] Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural Turing Machines. *CoRR* abs/1410.5401 (2014).

[12] Jiepu Jiang, Daqing He, and James Allan. 2014. Searching, Browsing, and Clicking in a Search Session: Changes in User Behavior by Task and over Time *(SIGIR'14)*. 607–616.

[13] Jaya Kawale, Hung H Bui, Branislav Kveton, Long Tran-Thanh, and Sanjay Chawla. 2015. Efficient Thompson Sampling for Online Matrix-Factorization Recommendation *(NIPS'15)*. 1297–1305.

[14] Andreas Krause and Cheng S. Ong. 2011. Contextual Gaussian Process Bandit Optimization *(NIPS'11)*. 2447–2455.

[15] John Langford and Tong Zhang. 2008. The Epoch-Greedy Algorithm for Multi-armed Bandits with Side Information *(NIPS'08)*. 817–824.

[16] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A Contextual-bandit Approach to Personalized News Article Recommendation *(WWW'10)*. 661–670.

[17] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. 2011. Unbiased Offline Evaluation of Contextual-bandit-based News Article Recommendation Algorithms *(WSDM'11)*. 297–306.

[18] Shuai Li, Alexandros Karatzoglou, and Claudio Gentile. 2016. Collaborative Filtering Bandits *(SIGIR'16)*. 539–548.

[19] Atsuyoshi Nakamura. 2015. A UCB-Like Strategy of Collaborative Filtering. In *Proceedings of the Sixth Asian Conference on Machine Learning (ACML'14)*. 315–329.

[20] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap. 2016. One-shot Learning with Memory-Augmented Neural Networks. *CoRR* abs/1605.06065 (2016).

[21] Rajat Sen, Karthikeyan Shanmugam, Murat Kocaoglu, Alexandros G. Dimakis, and Sanjay Shakkottai. 2016. Latent Contextual Bandits: A Non-Negative Matrix Factorization Approach. *CoRR* abs/1606.00119 (2016).

[22] Yilin Shen and Hongxia Jin. 2016. EpicRec: Towards Practical Differentially Private Framework for Personalized Recommendation *(CCS'16)*. 180–191.

[23] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias W. Seeger. 2010. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design *(ICML'10)*. 1015–1022.

[24] Michal Valko, Nathan Korda, Rémi Munos, Ilias Flaounas, and Nello Cristianini. 2013. Finite-time Analysis of Kernelised Contextual Bandits *(UAI'13)*. 654–663.

[25] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. 2016. Matching Networks for One Shot Learning *(NIPS'16)*. 3630–3638.

[26] Huazheng Wang, Qingyun Wu, and Hongning Wang. 2016. Learning Hidden Features for Contextual Bandits *(CIKM'16)*. 1633–1642.

[27] Qingyun Wu, Huazheng Wang, Quanquan Gu, and Hongning Wang. 2016. Contextual Bandits in a Collaborative Environment *(SIGIR'16)*. 529–538.

[28] Li Zhou and Emma Brunskill. 2016. Latent Contextual Bandits and Their Application to Personalized Recommendations for New Users *(IJCAI'16)*. 3646–3653.