MAMO: Memory-Augmented Meta-Optimization for Cold-start Recommendation

Manqing Dong
University of New South Wales
dongmanqing@gmail.com

Feng Yuan
University of New South Wales
Sydney, Australia

Lina Yao
University of New South Wales
lina.yao@unsw.edu.au

Xiwei Xu
Data 61, CSIRO
Sydney, Australia

Liming Zhu
Data 61, CSIRO
Sydney, Australia

ABSTRACT

A common challenge for most current recommender systems is the cold-start problem. Due to the lack of user-item interactions, the fine-tuned recommender systems are unable to handle situations with new users or new items. Recently, some works introduce the meta-optimization idea into the recommendation scenarios, i.e. predicting the user preference by only a few of past interacted items. The core idea is learning a global sharing initialization parameter for all users and then learning the local parameters for each user separately. However, most meta-learning based recommendation approaches adopt model-agnostic meta-learning for parameter initialization, where the global sharing parameter may lead the model into local optima for some users. In this paper, we design two memory matrices that can store task-specific memories and feature-specific memories. Specifically, the feature-specific memories are used to guide the model with personalized parameter initialization, while the task-specific memories are used to guide the model fast predicting the user preference. And we adopt a meta-optimization approach for optimizing the proposed method. We test the model on two widely used recommendation datasets and consider four cold-start situations. The experimental results show the effectiveness of the proposed methods.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

Recommender systems; Cold-start problem; Meta learning

1 INTRODUCTION

Personalized recommender systems are playing more and more important roles in web and mobile applications. Despite the success of traditional matrix-factorization based recommendation methods [4] or the most current deep-learning based techniques [33], a common challenge for most recommendation methods is the cold-start problem [28]. Because of the lack of user-item interactions, the recommendation approaches that utilize such interactions are unable to handle the situations where new users or items exist, corresponding to the user cold-start and item cold-start problems.

Traditional way to solve the cold start problem is leveraging auxiliary information into the recommender systems, e.g.,content-based recommender systems [21, 27] and cross-domain recommenders [16, 25]. For example, to address the item cold-start problem, [27] proposes a hybrid model in which item features are learned from the descriptions of items via a stacked denoising autoencoder and further combined into a collaborative filtering model timeSVD++. In [25], the authors propose a cross-domain latent feature mapping model, where the neighborhood-based cross-domain latent feature mapping method is applied to learn a feature mapping function for each cold-start user. However, the major limitation of such approaches is that the learned model may recommend same items for users with similar content thereby neglect the personal interests.

Inspired by recent works in few-shot learning [26], meta-learning [24] has been introduced into the recommender systems to solve the cold-start problem, i.e. predicting a user's preferences by only a few past interacted items. Most of the current meta-learning based recommender systems [5, 15, 34] adopt optimization-based algorithms such as model-agnostic meta-learning (MAML) [11], for their promising performance in learning configuration initialization for new tasks. Generally, the recommendation for a user is regarded as a learning task. The core idea is learning a global parameter to initialize the parameter of personalized recommender models. The personalized parameter will be locally updated to learn a specific user's preference, and the global parameter will be updated by minimizing the loss over the training tasks among the users. Then, the learned global parameter is used to guide the model settings for new users. For example, [15] uses several fully connected neural networks as recommender models. They first learn user and item embedding from the user and item profiles, and then feed them

into the recommender model to get predictions. They define two groups of global parameters: parameters in learning embeddings and parameters in recommender models. They locally update the recommender model for personalized recommendation and globally update the two groups of parameters for the initialization of new users. [8] also constructs a deep neural net as the recommender model. They define a global parameter for initializing the recommender model, and then locally updates the model parameter by introducing update and stop controllers.

The aforementioned approaches have been proven to be promising with regards to applying meta-optimization in the cold-start scenarios and presenting competitive performance in warm-start scenarios. However, they have the following limitations. Most of them are build upon MAML algorithm and its variants, which are powerful to cope with data sparsity, while having a variety of issues such as instability, slow convergence, and weak generalization. In particular, they often suffer from gradient degradation ending up with a local optima when handling users who show different gradient descent directions comparing with the majority of users in the training set. To address this issue, we propose Memory-Augmented Meta-Optimization (MAMO) for cold-start recommendation. In details, **1)** for solving the problem of local optima, we design a feature-specific memory to provide a personalized bias term when initializing the model parameters. Specifically, the feature-specific memory includes two memory matrices: one stores the user profile memory to provide retrievable attention values, and the other caches the fast gradients of the previous training sets to be read according to the retrievable attention values. **2)** We further design a task-specific memory cube, i.e. user preference memory, to learn to capture the shared potential user preference commonality on different items. It is used as fast weights for the recommender model to alleviate a need for storing copies of neural activity patterns. **3)** The extensive experiments on two widely used datasets show MAMO performs favorably against the state-of-the-arts. The code is publicly available for reproduction[1].

The rest of the paper is organized as follows. We review proposed approach in Section 2; Section 3 presents the settings, experimental results and analysis of the experiments; we review the related work in Section 4, followed by the conclusion in Section 5.

2 PROPOSED APPROACH

2.1 Overview

2.1.1 Problem Definition. We consider the recommendation for a user as one task. Given a user u with profile p_u and rated items $\mathcal{I}_u^S$, where each item i is associated with a description file p_i and corresponding ratings $y_{u,i}^S$, our goal is predicting the rating $y_{u,i}^Q$ by user u for a new item $i \in \mathcal{I}_u^Q$. Here, S stands for the support set, and Q stands for the query set.

2.1.2 Motivation. Most existing meta learning based recommendation techniques attempt to learn a meta global parameter ϕ to guide the model initialization $\mathcal{R}_\theta$, i.e. $\theta \leftarrow \phi$, where ϕ is learned across all the users in training set. It provides a uniform parameter initialization to govern the trained recommendation model
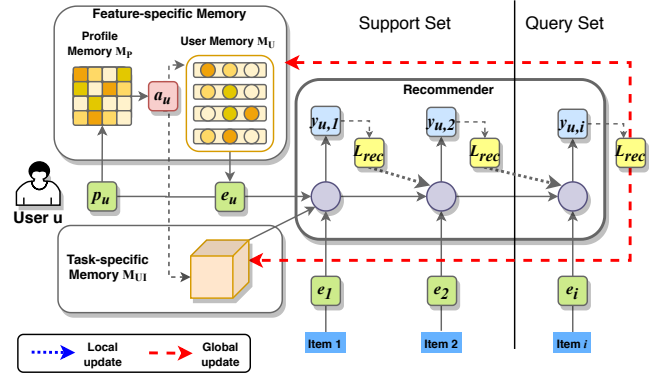
Figure 1: The training phase of MAMO

to predict on new users. The global parameter ϕ works uniformly across all users and thus may be inadequate to discern the intrinsic discrepancies among a variety of user patterns, resulting in poor generalization. Also, the model may tend to be stuck on a local optimum and encounter gradient degradation. Instead of learning a single initialization of the model parameters, we propose an adaptive meta learning recommendation model, Memory-Augmented Meta-Optimization (MAMO), to improve the stability, generalization and computational cost of model by learning a multi-level personalized model parameters.

2.1.3 Model Structure. The proposed model includes two parts: the recommender model for predicting the user preference and the memory-augmented meta-optimization learner for initializing recommender model parameters. The recommender model $\mathcal{R}_\Theta(p_u, p_i)$ predicts the recommendation scores, where the model parameters $\Theta = \{\theta_u, \theta_i, \theta_r\}$ will be locally updated for single users to provide personalized recommendation. The meta-learner, which includes global parameters $\Phi = \{\phi_u, \phi_i, \phi_r\}$ and memories $M = \{M_U, M_P, M_{U,I}\}$, will provide personalized initialization for the recommender model parameters Θ, and will be globally updated during the training process for users $u \in U^{train}$. The learned meta-learner is further used to initialize the recommender model for new users $u \in U^{test}$.

2.2 Recommender Model

Similar to many previous works, we assume that the user preference is coming from a complex combination of his/her personal information such as user profiles, user rating records, and the item profiles.

2.2.1 Embedding: $p_u \rightarrow e_u$, $p_i \rightarrow e_i$. We use user profile p_u to represent the initial user preference for the following considerations. First, user profile normally includes general information such as age groups and locations. In a cold-start scenario, where there are only limited user-item interaction records, this will provide potential user preference for the recommenders. Second, traditional one-hot user representation (by a unique id) is strongly relied on collaborative filtering techniques. The fine-tuned model is hard to adapt to new users. Similarly, we use p_i to denote the item profile for learning the item embedding.

To learn the user and item embedding vectors $e_u \in \mathbb{R}^{d_e}$ and $e_i \in \mathbb{R}^{d_u}$ from $p_u \in \mathbb{R}^{d_u}$ and $p_i \in \mathbb{R}^{d_i}$, the simplest way is constructing a multi-layer fully connected neural network, i.e.

$$e_u = f_{\theta_u}(p_u), e_i = f_{\theta_i}(p_i) \tag{1}$$

where d_e is the embedding size; f denotes the fully connected layers; d_u and d_i stand for the dimension of the user profile vector and item profile vector; θ_u and θ_i are the parameters of the fully connected layers for learning user and item embedding, respectively. We will omit similar notations in the rest of the paper.

2.2.2 Recommendation: $(e_u, e_i) \rightarrow y_{u,i}$.
Given the user embedding e_u, and a list of rated item embeddings e_i for $i \in \mathcal{I}_u^Q$, we get the prediction of preference score $y_{u,i}$ for each item by:

$$\hat{y_{u,i}} = f_{\theta_r}(e_u, e_i) = FC_{\theta_r}(M_{u,I} \cdot [e_u, e_i]) \tag{2}$$

where $[e_u, e_i]$ is the concatenation of the user embedding and the item embedding, and FC denotes the fully connected layers. $M_{u,I} \in \mathbb{R}^{d_e \times 2d_e}$ is a matrix that includes the fast weights of the recommender model for user u, which is extracted from the task-specific memory $M_{U,I}$ according to the user profile p_u, i.e. $M_{u,I} \leftarrow (p_u, M_{U,I})$. The task-specific memory $M_{u,I}$ for user u, will be locally updated (i.e. during the learning on support set of user u) for personalized recommendation. We will introduce more details about the task-specific memory in the further parts.

2.3 Memory-Augmented Meta-Optimization

2.3.1 Feature-specific Memory: $(p_u, M_U, M_P) \rightarrow b_u$.
Recall that the parameters used for extracting user and item embedding are θ_u and θ_i, traditional meta optimization approach will learn the global parameters ϕ_u and ϕ_i for initialization, i.e. $\theta_u \leftarrow \phi_u, \theta_i \leftarrow \phi_i$. Here for addressing the single initialization problem, we introduce feature-specific memories, i.e. the user embedding memory M_U and the profile memory M_P, to help the model initialize a personalized parameter θ_u. The profile memory M_P stores information relevant to user profiles p_u to provide the retrieval attention values a_u. The retrieval attention values are applied for extracting information from user embedding memory M_U, where each row of M_U keeps the according fast gradients (or bias terms). Together, the two memory matrices will help to generate a personalized bias term b_u when initializing θ_u, i.e. $\theta_u \leftarrow \phi_u - \tau b_u$. Here the bias term b_u can be regarded as a personalized initialization parameter for guiding the global parameter ϕ_u to fast adapt to the case of user u; the τ is a hyper-parameter for controlling how much bias term is considered when initializing θ_u.

In details, given a user profile p_u, the profile memory $M_P \in \mathbb{R}^{K \times d_u}$ will calculate the attention values $a_u \in \mathbb{R}^K$ by

$$a_u = attention(p_u, M_P) \tag{3}$$

where $attention$ function calculates the cosine similarity between the user profile and the user profile memory, and then be normalized by softmax function. The dimension K denotes the number of user preference types, which is a predefined number before the training process. We obtain the personalized bias term b_u by

$$b_u = a_u^\top M_U \tag{4}$$

where $M_U \in \mathbb{R}^{K \times \{d_{\theta_u}\}}$ stores the memory of the fast gradients. Notice that the user embedding model may comprise more than

one neural layer and more than one parameter, which means M_U is not a numerical matrix but stores all the fast gradients with the same shape as the parameters in the user embedding layers, so here d_{θ_u} denotes the dimension of the parameters in user embedding layers.

Before the training process, the two memory matrices are randomly initialized, and will be updated during the training process. Specifically, profile memory will be updated by:

$$M_P = \alpha \cdot (a_u p_u^\top) + (1 - \alpha)M_P \tag{5}$$

where $(a_u p_u^\top)$ is the cross product of a_u and p_u, α is a hyper-parameter to control how much new profile information is added. Here we add an attention mask a_u when add the new information so that the new profile information will be attentively added to the memory matrix. Similarly, the parameter memory M_U will be updated by

$$M_U = \beta \cdot (a_u \nabla_{\theta_u}(\mathcal{L}(\hat{y_{u,i}}, y_{u,i}))) + (1 - \beta)M_U \tag{6}$$

where $\mathcal{L}(\hat{y_{u,i}}, y_{u,i})$ denotes the training loss, and β is the hyper-parameter to control how much new information is kept.

2.3.2 Task-specific Memory: $(a_u, M_{U,I}) \rightarrow M_{u,I}$.
The user preference matrix $M_{u,I} \in \mathbb{R}^{d_e \times 2d_d}$ serves as fast weights or a transform matrix for the recommender model from the user and item embedding (see equation (2)) that is extracted from the memory cube $M_{U,I} \in \mathbb{R}^{K \times d_e \times 2d_e}$, where K is the same notation for the dimension of feature-specific memory that denotes the number of user preference type. Similar to the feature-specific memory, which follows the idea of Neural Turing Machine (NTM)[13], the memory cube $M_{U,I}$ have a read head to retrieve the memory and a write head to update the memory. Similarly, we attentively retrieve the preference matrix $M_{u,I}$ from $M_{U,I}$ by:

$$M_{u,I} = a_u^\top \cdot M_{U,I} \tag{7}$$

where $a_u \in \mathbb{R}^K$ is learned from equation 3. The write head will write the updated personal preference memory matrix $M_{u,I}$ to the $M_{U,I}$ after the learning on the support set.

$$M_{U,I} = \gamma \cdot (a_u \otimes M_{u,I}) + (1 - \gamma)M_{U,I} \tag{8}$$

where $\otimes$ denotes the tensor product, γ is a hyper-parameter to control how much new preference information is added.

2.3.3 Local Update.
Traditionally, the parameters of a neural network are initialized by randomly sampling from a statistical distribution. Given sufficient training data, the randomly initialized parameters can usually converge to a good local optimum but may take a long time [8]. In the cold-start scenario, random initialization combined with limited training data can lead to serious over-fitting, which makes the trained recommender insufficient to generalize well. Inspired by recent works of meta-training[15], we initialize the recommender parameters from the global initial values.

At the beginning of the training process, we randomly initialize the global parameters, i.e. $\phi_u, \phi_i, \phi_r, M_U, M_P, M_{U,I}$. For each user $u \in U^{train}$, we have support set and query set. During the local learning phase (i.e. learning on the support set), we initialize the local recommender parameters by:

$$\theta_u \leftarrow \phi_u - \tau b_u \tag{9}$$

$$\theta_i \leftarrow \phi_i, \theta_r \leftarrow \phi_r \tag{10}$$

Algorithm 1: Training process of MAMO

Input: Training user set U^{train}; User profile $\{p_u|u \in U^{train}\}$; Item profile $\{p_i|i \in (I_u^S, I_u^Q)\}$; User ratings $\{y_{u,i}|u \in U^{train}, i \in (I_u^S, I_u^Q)\}$; Hyper-parameters $\alpha, \beta, \gamma, \tau, \rho, \lambda$

Output: Meta parameters $\phi_u, \phi_i, \phi_r, M_P, M_U, M_{U,I}$

1 Randomly initialize the meta parameters ϕ_u, ϕ_i, ϕ_r;
2 Randomly initialize the memories $M_P, M_U, M_{U,I}$;
3 **while** *Not Done* **do**
4   **for** $u \in U^{train}$ **do**
5   Calculate bias term $b_u \leftarrow (p_u, M_U, M_P)$ by Eq. (3-4);
6   Initialize the local parameters $\theta_i, \theta_u, \theta_r$ by Eq. (9-10);
7   Initialize the preference memory $M_{u,I} \leftarrow (a_u, M_{U,I})$ by Eq. (7);
8   **for** $i \in I_u^S$ **do**
9   Get user and item embedding e_u and e_i by Eq. (1);
10   Get prediction of $\hat{y_{u,i}}$ by Eq. (2);
11   Local update $M_{u,I}$;
12   Local update $\theta_u, \theta_i, \theta_r$ by: $\theta_* \leftarrow \theta_* - \rho \cdot \nabla_{\theta_*} \mathcal{L}(y_{u,i}, \hat{y_{u,i}})$;
13   **end**
14   **end**
15   Update feature-specific memory M_P, M_U by Eq. (5-6);
16   Update task-specific memory $M_{U,I}$ by Eq. (8);
17   Update global parameters ϕ_u, ϕ_i, ϕ_r by: $\phi_* \leftarrow \phi_* - \lambda \Sigma_{u \in U^{train}} \Sigma_{i \in I_u^Q} \nabla \mathcal{L}(\mathcal{R}_{\hat{\theta}_*})$;
18 **end**

---

where b_u is obtained via equation (3-4). Then we obtain the task-specific memory matrix $M_{u,I}$ by equation (7). The prediction of ratings for items $i \in I_u^S$ is based on equation (2). The optimization goal in local training is to minimize the loss of the recommendation for a single user, i.e. updating the local parameters by minimizing the prediction loss $\mathcal{L}(y_{u,i}, \hat{y_{u,i}})$. Thus, the local parameters will be updated by:

$$\theta_* \leftarrow \theta_* - \rho \cdot \nabla_{\theta_*} \mathcal{L}(y_{u,i}, \hat{y_{u,i}}) \tag{11}$$

where $*$ could be either i, u, or r; ρ is the learning rate for updating the local parameters. The preference matrix $M_{u,I}$ will also be locally updated via back-propagation.

2.3.4 Global Update. The aim of the meta optimization is to minimize the expected loss on the local query set $i \in I_u^Q$ for $u \in U^{train}$. Here, the parameters of the meta-learner include: shared initial parameters ϕ_u, ϕ_i and ϕ_r; feature-specific memories, i.e. profile memory M_P and user embedding memory M_U; and the task-specific memory $M_{U,I}$. The gradients related to the meta-testing loss, which we call meta-gradient, can be computed via back-propagation. The meta-gradient may involve higher-order derivatives, which are expensive to compute when the depth of the neural nets is deep. Therefore, MAML[22] takes one-step gradient descent for meta-optimization. We take similar ideas, where after the local training

on the support set, we update the global parameters according to the loss on query sets.

Suppose the recommender model is denoted as $\mathcal{R}_\theta$ combined with a task-specific memory $M_{u,I}$ for user u, where $\theta = \{\theta_u, \theta_i, \theta_r\}$. After the local training on support set, we get the model $\mathcal{R}_{\hat{\theta}}$ with updated parameters $\hat{\theta}$. Our goal is minimizing the training loss for users $u \in U^{train}$ on query sets for $i \in I_u^Q$. Then, the global parameters are updated by

$$\phi_* \leftarrow \phi_* - \lambda \Sigma_{u \in U^{train}} \Sigma_{i \in I_u^Q} \nabla \mathcal{L}(\mathcal{R}_{\hat{\theta}_*}) \tag{12}$$

Meanwhile, the feature-specific memories M_P and M_U will be updated by equation (5) and (6); the task-specific memory $M_{U,I}$ will be updated by equation (8). The pseudo code of the training process is listed in algorithm 1.

3 EXPERIMENTS

3.1 Datasets

3.1.1 Datasets. We use two widely used public available datasets for evaluation: MovieLens 1M[2] and Book-crossing[3], which have both user information and item information. MovieLens 1M includes around 1 million ratings from about 6 thousand users for over 3 thousand movies, and the ratings range from 1 to 5. For the Book-crossing dataset, we filter the records that users or items do not have relevant profiles. The processed dataset includes about 600 thousand ratings from around 50 thousand users for over 50 thousand books, the ratings are between 1 and 10. See appendix A.1 for more details.

3.1.2 Data preprocessing. We use different strategies when learning the feature representation. For features have single categorical value, such as location and occupation, each feature is denoted by index and is represented with a randomly initialized embedding vector; we use similar approaches to process the numerical features such as the publication year and age group. For features that may have multiple categories, such as movie genres and directors, we use one-hot representation and then transform them into vectors that have same dimension with other feature vectors.

3.1.3 Training and testing dataset. Each user is regarded as a sample in the dataset. We randomly separate the users into training and testing users with ratio 80:20. The history records for a single user are further divided into support set and query set. We trim the number of rating records for each user as with 20 records to force the model to learn from few samples. In default settings, we consider the first 15 items for a user as the support set and the others as the query set. A detail is that we sort these items according to the user review time. By doing so, the rated items in query set are logically regarded as 'new items' for a user. We will further discuss whether the ratio of training dataset or the number of cases in the support set will affect the model performance.

3.1.4 Cold-start scenarios. We further consider the recommendation performance on four scenarios: 1) existing users for existing items (W-W) ; 2) existing users for cold items (W-C); 3) cold users for existing items (C-W); and 4) cold users for cold items (C-C).

---

[2]https://grouplens.org/datasets/movielens/
[3]http://www2.informatik.uni-freiburg.de/~cziegler/BX/

For the MovieLens dataset, we classify the users into warm or cold users according to their first comment time. The first comment time in the MovieLens dataset ranges from 2000-04-26 to 2003-03-21. We found most users (about 90%) provide their first comments before 2000-12-03. By dividing the users based on this time, we have 5,400 warm users and 640 cold users. As for the items, we regard items with less than 10 ratings as cold items, where we get 1,683 warm items and 1,645 cold items. For the Book-crossing dataset, where the rating time is not provided, we assume the order of the user ids is consistent with the order of the user registration time. Thus, we assume the first 90 percent of the users are warm users and the others are cold users. Similarly, we regard items with more than 5 ratings as warm items, from which we get 628 warm items and the others as cold items.

3.2 Parameter Studies

Here we show the parameter studies on several key parameters of our method: the model set-up parameters when initializing the training and testing dataset; the model structure parameters for constructing the recommender models; and the hyper-parameters during the learning process. We present the results under two evaluation metrics, i.e. Mean Absolute Error (MAE) and normalized discounted cumulative gain ($NDCG$):

$$MAE = \frac{1}{|U^{test}|} \Sigma_{u \in U^{test}} \frac{1}{|\mathcal{I}_u^Q|} \Sigma_{i \in \mathcal{I}_u^Q} |y_{u,i} - \hat{y_{u,i}}| \qquad (13)$$

$$NDCG@N = \frac{1}{|U^{test}|} \Sigma_{u \in U^{test}} \frac{DCG@N}{IDCG@N} \qquad (14)$$

$$DCG@N = \Sigma_{n=1}^{N} \frac{2^{y_{u,n}} - 1}{\log(n+1)} \qquad (15)$$

where N is the number of instances in the query set for each user; $DCG@N$ calculates the $top - N$ actual rating values sorted by predicted rating values; and $IDCG@N$ calculates the $top - N$ sorted actual rating values, i.e. the most possible values, for $DCG@N$. MAE evaluates the accuracy of rating prediction, where a lower value indicates better model performance. $NDCG@N$ considers the preference ordering performance on observed query sets, where a higher value indicates a better performance. The experiments are conducted on MovieLens dataset.

3.2.1 Model set-up parameters. Here we consider the model performance under different separating ratios for training-testing or support-query sets. The experimental results (Figure 2 (a)) suggest a higher ratio of the training set improves the model performance; while the performance for training with only half of the users is still acceptable, which indicates the effectiveness of the meta-training approach. As for the sample size of the support set, where we control the maximum of rating records for a user as 20, we compare the results with support sample size between 5 to 15, where we use the last 5 samples for each user as the query sample set for a fair comparison. According to Figure 2 (b), we can find that a larger sample size provides better results, while the model can also provide acceptable predictions with smaller sample size, which shows its capacity in cold scenarios.

3.2.2 Model structure parameters. The settings of the parameters for deep neural nets, especially the number of layers and the dimension of layer nodes, can sometimes affect the model performance. Generally, a network with large dimension and a complex structure may provide accurate predictions but is data hungry and slow in convergence. In our work, we use fully connected layers to learn the user and item embeddings. According to Figure 2 (c)-(d), which show the model performance under different number of embedding layers and embedding size d_e, we could see that the number of neural layers has slight impacts on the model performance, while too shallow or too deep layers may lead to bad results. A moderate setting of the embedding size can provide acceptable prediction results and require less training time.

3.2.3 Hyper-parameters. We have hyper-parameters α and β (ranging from 0 to 1) for globally updating the profile memory M_P and user embedding memory M_U in feature-specific memory, and γ for globally updating the task-specific memory $M_{U,I}$. The hyper-parameters control how much new information is added to the memory matrices. Figure 2 (e)-(f) show the parameter sensitivity for parameter α and β. For parameter α, values around 0.5 provide better performance of the models, because of the update strategy of the memory matrices. Since the profile memory is randomly initialized and is not updated by back-propagation, the profile memory requires more new information to complete the distribution of the user profiles. While for the user embedding memory, too much new information may cause chaos in existing memory, where $\beta = 0.1$ provides the best performance. The parameter γ for task-specific memory (not listed in the figures) has similar patterns with β, where it provides best results with values around 0.1. We have ρ and λ for updating local or global parameters, and τ for providing personalized bias term. Figure 2 (g)-(h) show the results for ρ and λ, where small learning rates provide best results, since a large value may make the model difficult in convergence. As for the parameter τ, which is not listed in the figures, we find the values around 0.1 perform best.

3.3 Comparison Results

3.3.1 Comparison Methods. We adopt the following representative state-of-the-arts that apply the meta-learning idea into the recommender systems, where the first three methods are based on meta-optimization, and the last method, which includes item-level model and feature-level model, is based on memory networks.

- **MeLU** [15]: get rating predictions by feeding the concatenation of user and item embeddings into fully connected layers. The local parameter of the fully connected layers will be locally updated for personalized recommendation. The global parameter of the recommender and the parameters of embedding layers will be globally updated for all the users.
- **MetaCS-DNN** [3]: follows a similar idea of MeLU when constructing the recommender model, i.e. several fully connected layers, the difference is the local and global parameters involve all parameters from the inputs to the predictions.
- s^2 **Meta** [8]: uses a deep neural net as the recommender model and updated via meta-optimization approach. Specifically, it introduced update and stop controllers to update the local parameters.
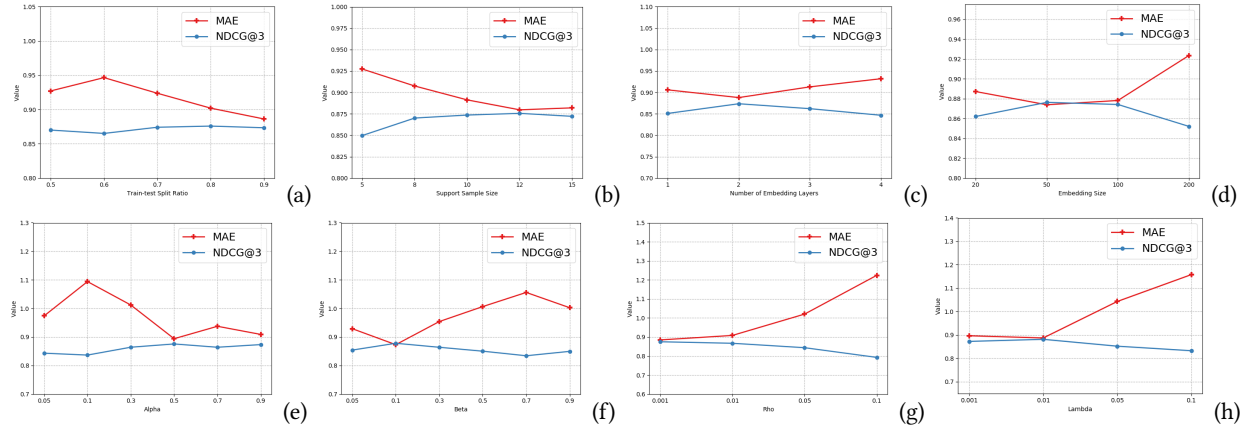
Figure 2: Parameter sensitivity in different settings.

Table 1: Comparison Results

Method	Metrics	MovieLens-1M				Book-crossing			
		W-W	W-C	C-W	C-C	W-W	W-C	C-W	C-C
MeLU	MAE	0.7661	0.9361	0.7884	0.9299	0.7799	2.1047	1.8701	2.1475
	$NDCG@3$	0.8904	0.7990	0.8810	0.8011	0.9572	0.8441	0.8527	0.8410
MetaCS-DNN	MAE	0.9047	1.1694	1.0625	1.2012	1.6206	2.1457	2.2648	2.3088
	$NDCG@3$	0.9090	0.7715	0.8559	0.7680	0.8860	0.8365	0.8202	0.8160
s^2 Meta	MAE	0.7567	0.8860	0.8443	0.9130	1.5147	1.9767	1.8738	2.0921
	$NDCG@3$	0.8870	0.8323	0.8401	0.8148	0.8781	0.8463	0.8490	0.8422
Item-level RUM	MAE	0.8874	1.3424	1.2655	1.3509	1.4611	2.2857	2.2608	2.3197
	$NDCG@3$	0.9102	0.7692	0.7420	0.7680	0.8920	0.8265	0.8202	0.8160
Feature-level RUM	MAE	0.8739	1.2439	1.2488	1.2773	1.3945	2.0837	2.0322	2.1260
	$NDCG@3$	0.9120	0.7721	0.7642	0.7547	0.8909	0.8430	0.8338	0.8273
MAMO	MAE	0.8725	0.9306	0.8967	0.8894	1.4879	1.7379	1.6217	1.8188
	$NDCG@3$	0.8866	0.8315	0.8799	0.8709	0.8879	0.8402	0.8565	0.8384

- **RUM**: [6] uses neural matrix factorization as the recommender model, where the score for an item is the product of the user and item embeddings. The user embedding is learned from a user's intrinsic embedding and a memory embedding. The memory embedding is learned by either item-level or feature-level memories, i.e. **item-level RUM** and **feature-level RUM**. The item-level RUM stores and extracts the memory according to different items, while the feature-level RUM considers the similarity of the different features. Here, the memory matrix is initialized by each user's history records and is locally updated for personalized recommendation.

The parameter settings for our method and the comparison methods can be found in appendix A.2.

3.3.2 Comparison Results. The comparison results are listed in Table 1. Generally, we can see that the meta-optimization based methods show outstanding performance in cold scenarios, while RUM outperforms other methods in warm situations. This can be attributed to the stronger capability of meta-optimization approach

in capturing the general preferences. Since the memory matrices designed in RUM store only the personal history information, it needs more history records to establish an efficient memory matrix. Comparing MetaCS-DNN and MeLU, which utilize different strategies for updating embedding parameters, the results show that locally updating the embedding parameters performs better than globally updating the embedding parameters. We can see that our method shows stable and well performance in different scenarios, which shows the effectiveness of the meta-augmented meta-optimization strategy: learning a personalized initialization of local parameters from the memories.

3.4 Discussion

MAMO has two groups of memories. The first group includes user profile memory M_P and user embedding memory M_U that are utilized for providing personalized bias term when initializing the local parameters. The second group includes the task-specific memory $M_{U,I}$. In this section, we will discuss the impact of these two groups of memories. Also, we predefined K user types when constructing the aforementioned memories. We will discuss how to set
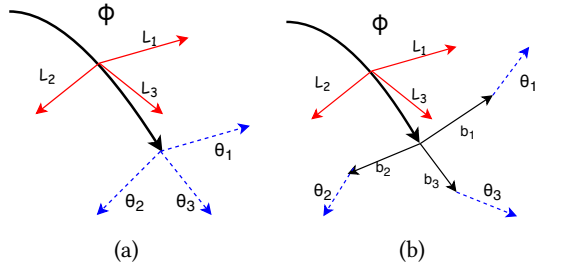
Figure 3: The major difference between MAML and our method. Red lines stand for the global/meta-optimization, the blue lines stand for the local learning/adaption.

(a)Training: |S|=10   (b)Testing: |S|=10

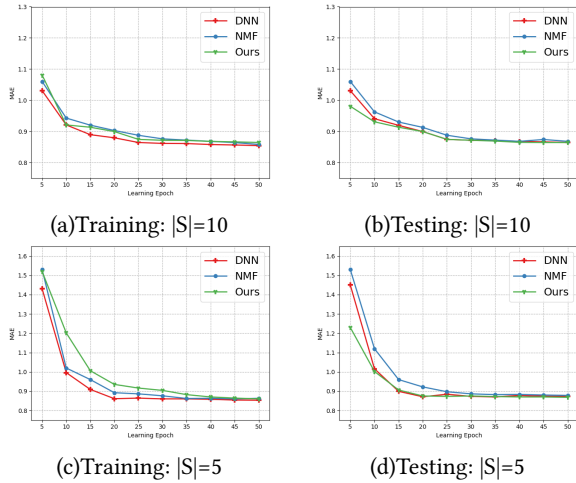(c)Training: |S|=5   (d)Testing: |S|=5

Figure 4: The performance during training process and testing process with different settings of support sample size on different recommender models.

an appropriate K and present an example of the standard users for K user preference patterns.

3.4.1 Impact of Personalized Bias Term. The basic idea of meta-optimization process is learning a sharing global parameter ϕ when initializing the personalized local parameter θ (see Figure 3 (a)). For addressing the potential local optima issues by the unique global sharing parameter for most current meta-optimization based works, we introduce two memory matrices to provide a personalized bias term b_u when initializing the local parameters θ_u when learning the user embedding. Specifically, the profile memory M_P stores K types of user profiles, and M_U stores corresponding bias terms. Then, the local parameters will be locally updated for personalized recommendation (see Figure 3 (b)). The comparison methods, MeLU and MetaCS-DNN, both utilize MAML for meta-optimization. According to the comparison results, we can see that our strategy outperforms these two methods in cold scenarios, especially for cold users. The reason may lie in that our optimization approach takes the user profile into consideration. The profile memory will automatically cluster users with similar profiles to provide fast bias terms.
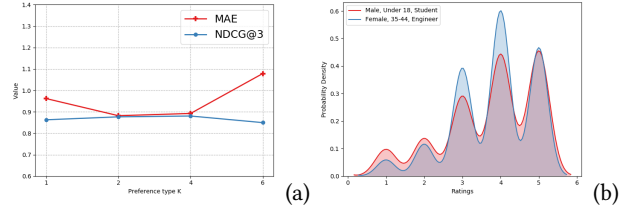

Figure 5: (a)Performance with different K. (b) Example for representative users when $K = 2$.

3.4.2 Impact of Preference Memory. We also introduce the task-specific memory $M_{U,I}$ that contains K types of preference matrices during the recommendation process. To see whether it helps the recommendation process, we compare it with different designs of recommender models. The first is predicting the ratings by the concatenation of the user and item embedding through several fully connected layers, denoted by DNN. The second is taking the idea of neural matrix factorization [14], which multiplies the user embedding and item embedding as the predictions, denoted by NMF. The results are shown in Figure 4, where (a), (c) show the performance on the query dataset during the training process; (b), (d) show the performance on the query dataset during the testing process. We can observe that DNN and NMF converge fast in the training process but perform unwell on testing dataset with small support sample size, while our method needs more steps to store the preference information but can learn fast during the testing process.

3.4.3 Discussion of Preference Type. In this work we predefined K user types when constructing the proposed memories, which can be regarded as the number of clusters of users based on the user profiles. Figure 5 (a) shows the model performance under different K. We can see that the K values around 2-4 achieve the best performance. A larger K has potential in providing more accurate guidance in recommendation, but it will take more computation cost (each type in M_U stores parameters with the same shape as all the network parameters in user embedding networks). In Figure 5 (b), we present the case when $K = 2$. We selected two representative user profiles which are most similar to M_P, the distributions are from the top 100 users that belong to these two types.

3.4.4 Limitations and Future work. In cold-start scenarios, auxiliary information is valuable for providing potential recommendations. Our work targets at the situations that users or items have limited interaction histories and utilizes the idea of few-shot learning to provide recommendations. We assume we have enough essential profile details of the users or items. During the data preprocessing process, we filter the data that are without relevant profiles. However, in the real-world cases, the auxiliary information is not always attainable, where our model becomes inefficient under such circumstance. In future works, we plan to leverage the knowledge from multi-modal studies [2] to address this issue. For example, Wu et al. [29] design a multimodal variational autoencoder that learns a joint feature distribution under any missing modalities.

4 RELATED WORK

4.1 Cold-start Problem in Recommender Systems

It is still a tricky and significant problem for recommenders to accurately recommend items to users. Some solutions in solving this problem have been provided in current publications [20], involving second-domain knowledge transfer [16], auxiliary and contextual information [21, 31], active learning [35] and deep learning [9].

Commonly, it is reported that the more information the better recommendation results are. [19] adds the matrix factorization with clusters to recommendations of cross-domains. Similarly, [16] presents an innovative model of cross-domain recommendation according to the partial least squares regression (PLSR) analysis. Both PLSR-CrossRec and PLSR-Latent can be utilized in rating of source-domain for better predicting ratings of cold-start users. Apart from leveraging auxiliary information from another domain, some researchers have regarded representative item content as auxiliary information to learn representative latent factor when coped with the cold-start problem. According to implied feedback, [21] presents an approach named visual-CLiMF to learn representative latent factors for cold start videos, where emotional aspects of items are incorporated into the latent factor representations of video contents. In order to better use content characteristics in further latent characteristics, [7] proposed a next-song recommender system for cold-start recommendation by mapping both learning and updating within spaces of the audio characteristic and the item latent. In addition, active learning has found its way in tackling both the user cold-start problem in both users [10] and the items [1]. [35] combines the active learning approach with item's attribute information to deal with the cold-start problem in items. More recent works take advantage of deep learning [9, 27, 32]. [18] proposes a multiplex interaction-oriented service recommendation approach (MISR), which merged different interactions into a deep neural network to search latent information so that new users can be better dealt with. [27] proposes a hybrid recommendation model in researching the content characteristics of items by means of a deep learning neural network and then exploited them to the timeSVD++ collaborative filtering model.

4.2 Meta-learning for Recommender Systems

Meta-learning, namely learning to learn, is learning across various tasks with fewer training samples in each task, which can be easily adapted to new tasks [24]. In recent years, meta-learning has been attracting much attention to alleviate cold-start problems [5, 15, 34], and most of them adopted optimization-based meta-learning approach and chose model-agnostic meta-learning (MAML) [11] for model training. Generally, those works define a recommendation model $\mathcal{R}_\theta$ with parameter θ, and a meta-learner $\mathcal{M}(\phi)$. Modeling each user's preference is regarded as a learning task, and the users are divided into training and testing users. The initialization of parameter θ is defined by some function $\theta \leftarrow \mathcal{M}(\phi)$, e.g. in a simplest case, $\theta \leftarrow \phi$. During the training process, the parameter of the recommendation model, i.e. θ, will be locally updated by minimizing the recommendation loss $\mathcal{L}(\mathcal{R}_\theta(D^{support}))$ on the support set $D^{support}$ (the history ratings by a user) for a single user; and

then used to retrieve the recommendation loss $\mathcal{L}(\mathcal{R}_\theta(D^{query}))$ on the query set D^{query}. The parameter of the meta-learner will be globally updated by minimizing the sum of recommendation loss $\Sigma_{u \in U^{train}}(\mathcal{L}(\mathcal{R}_{\theta \leftarrow \phi}(D^{query})))$ for all users in the training dataset. Then, the meta-learner will guide the learning process for new users by providing the initialization of parameter θ, which can expedite the learning process of the recommendation models for testing users.

The major differences among the above works lie in the design of the recommender model $\mathcal{R}_\theta$ and the meta-optimization approach $\mathcal{M}(\phi)$. For example, [15] targets at a rating prediction problem where they feed the concatenation of user and item embedding into fully-connected layers to get predictions. They locally update the parameters of neural layers to get personalized recommendation and globally update the parameters of embedding learning and a global parameter of neural layers. [8] constructs a deep neural nets as the recommender model, and they design an input gate and a forget gate during updating the local parameters. One limitation of the current meta-optimization approach for recommendation is that most works learn a shared global initialization parameter, which means the initialization parameter is same for all users. This may lead the recommender model to the local optima problem and to be slow in convergence. To address this issue, our work introduces two memory matrices to provide personalized bias terms when initializing the parameters.

4.3 Memory-augmented Neural Networks

With the ability to express, store, and manipulate the records explicitly, dynamically, and effectively, external memory networks [23] have gained popularity in recent years in fields such as question-answering systems [30] and knowledge tracking [12]. One branch of the memory neural networks is based on Neural Turing Machine (NTM) [13]: the memory is normally stored in a matrix with a read head to extract the information from the memory and a write head to update the memory at each time step. Such features make the NTM a good module for meta-learning or few-shot learning. Following this idea, [22] proposes Memory-augmented Neural Networks (MANN). Similar to NTM, they design a read controller to learn a key vector v_t from the inputs at time t, and v_t is used to learn the rating weights w_t^r for the rows in the memory matrix M_t; the retrieved vector is the sum of rows in memory matrix with the above attention values, i.e. $\Sigma_k w_t^r(k)M_t(k)$. Then, for updating the memory, the authors propose usage weights w_t^u to evaluate the usage of the rows in the memory matrix. The writing weights w_t^w are learned from the previous reading weights and the usage weights. Last, the memory is updated by $M_t(k) \leftarrow M_{t-1}(k) + w_t^w(k)v_t$.

The work by [6] is one of the first works that integrate MANN into recommendation tasks. They consider the matrix factorization model as a neural network, where the score $y_{u,i}$ for an item i from user u is predicted from the user and item embeddings, i.e. $y_{u,i} = e_u^\top e_i$. They propose memory enhanced the user embedding, where the user embedding is learned from a user's intrinsic embedding e_u^* and a memory embedding $e_u^M = read(M_u, e_i)$, i.e. $e_u \leftarrow f(e_u^*, e_u^M)$. The memory embedding e_u^M reads from the user personalized memory matrix M_u to retrieve the related information for item i according to the item embedding e_i. Another work by

[17] adds a gate mechanism to control how much past memory and current memory are kept. Different from these works where the memory matrices are designed for each user and cannot share among different users, we propose two global sharing memories: feature-specific memory and task-specific memory. The feature-specific memory includes user profile memory and user embedding memory. Given a user, this memory will extract the parameter memory according to the similarity between the user profile and the user profile memory. The task-specific memory stores the preference information, which serves as fast weights for the recommender model, to alleviate the need for storing copies of neural activity patterns.

5 CONCLUSION

A common challenge for most current recommender systems is the cold-start problem. Recently, inspired by the progress of few-shot learning, some works leverage the meta-optimization idea into the recommendation. The core idea is learning a global sharing initialization parameter of the recommender model for all users and then updating the local parameters to learn personalized recommender model for single users. A common limitation for most current works is that the global sharing initialization parameter is unique for all users, which may lead the recommendation model into local optima for users with distinct preference patterns. For addressing this issue, in this work, we design two memory matrices to provide personalized initialization for recommender models: feature-specific memory that provides a personalized bias term when initializing the recommender model, and a task-specific memory to guide the recommendation process. We evaluate the proposed methods on two public available datasets and provide details of the implementation. The experimental results show the effectiveness of our method.

REFERENCES

[1] Oren Anava, Shahar Golan, Nadav Golbandi, Zohar Karnin, Ronny Lempel, Oleg Rokhlenko, and Oren Somekh. 2015. Budget-constrained item cold-start handling in collaborative filtering recommenders via optimal design. In *Proceedings of the 24th International Conference on World Wide Web*. 45–54.

[2] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. 2018. Multimodal machine learning: A survey and taxonomy. *IEEE transactions on pattern analysis and machine intelligence* 41, 2 (2018), 423–443.

[3] Homanga Bharadhwaj. 2019. Meta-Learning for User Cold-Start Recommendation. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[4] Dheeraj Bokde, Sheetal Girase, and Debajyoti Mukhopadhyay. 2015. Matrix factorization model in collaborative filtering algorithms: A survey. *Procedia Computer Science* 49 (2015), 136–146.

[5] Fei Chen, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. 2018. Federated meta-learning for recommendation. *arXiv preprint arXiv:1802.07876* (2018).

[6] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In *Proceedings of the eleventh ACM international conference on web search and data mining*. ACM, 108–116.

[7] Szu-Yu Chou, Yi-Hsuan Yang, Jyh-Shing Roger Jang, and Yu-Ching Lin. 2016. Addressing cold start for next-song recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*. 115–118.

[8] Zhengxiao Du, Xiaowei Wang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Sequential Scenario-Specific Meta Learner for Online Recommendation. *arXiv preprint arXiv:1906.00391* (2019).

[9] Travis Ebesu and Yi Fang. 2017. Neural Semantic Personalized Ranking for item cold-start recommendation. *Information Retrieval Journal* 20, 2 (2017), 109–131.

[10] Mehdi Elahi, Francesco Ricci, and Neil Rubens. 2016. A survey of active learning in collaborative filtering recommender systems. *Computer Science Review* 20 (2016), 29–50.

[11] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1126–1135.

[12] Marjan Ghazvininejad, Chris Brockett, Ming-Wei Chang, Bill Dolan, Jianfeng Gao, Wen-tau Yih, and Michel Galley. 2018. A knowledge-grounded neural conversation model. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

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

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[15] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. 2019. MeLU: Meta-Learned User Preference Estimator for Cold-Start Recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1073–1082.

[16] Cheng-Te Li, Chia-Tai Hsu, and Man-Kwan Shan. 2018. A Cross-Domain Recommendation Mechanism for Cold-Start Users Based on Partial Least Squares Regression. *ACM Transactions on Intelligent Systems and Technology (TIST)* 9, 6 (2018), 1–26.

[17] Yunxiao Li, Jiaxing Song, Xiao Li, and Weidong Liu. 2019. Gated Sequential Recommendation with Dynamic Memory Network. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[18] Yutao Ma, Xiao Geng, and Jian Wang. 2020. A Deep Neural Network With Multiplex Interactions for Cold-Start Service Recommendation. *IEEE Transactions on Engineering Management* (2020).

[19] Nima Mirbakhsh and Charles X Ling. 2015. Improving top-n recommendation for cold-start users via cross-domain information. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 9, 4 (2015), 1–19.

[20] Nitin Mishra, Vimal Mishra, and Saumya Chaturvedi. 2017. Tools and techniques for solving cold start recommendation. In *Proceedings of the 1st International Conference on Internet of Things and Machine Learning*. 1–6.

[21] Sujoy Roy and Sharath Chandra Guntuku. 2016. Latent factor representations for cold-start video recommendation. In *Proceedings of the 10th ACM conference on recommender systems*. 99–106.

[22] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. 2016. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*. 1842–1850.

[23] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*. 2440–2448.

[24] Joaquin Vanschoren. 2018. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548* (2018).

[25] Xinghua Wang, Zhaohui Peng, Senzhang Wang, S Yu Philip, Wenjing Fu, Xiaokang Xu, and Xiaoguang Hong. 2019. CDLFM: cross-domain recommendation for cold-start users via latent feature mapping. *Knowledge and Information Systems* (2019), 1–28.

[26] Yaqing Wang and Quanming Yao. 2019. Few-shot learning: A survey. *arXiv preprint arXiv:1904.05046* (2019).

[27] Jian Wei, Jianhua He, Kai Chen, Yi Zhou, and Zuoyin Tang. 2016. Collaborative filtering and deep learning based hybrid recommendation for cold start problem. In *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 874–877.

[28] Jian Wei, Jianhua He, Kai Chen, Yi Zhou, and Zuoyin Tang. 2017. Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications* 69 (2017), 29–39.

[29] Mike Wu and Noah Goodman. 2018. Multimodal generative models for scalable weakly-supervised learning. In *Advances in Neural Information Processing Systems*. 5575–5585.

[30] Caiming Xiong, Stephen Merity, and Richard Socher. 2016. Dynamic memory networks for visual and textual question answering. In *International conference on machine learning*. 2397–2406.

[31] Lina Yao, Quan Z Sheng, Xianzhi Wang, Wei Emma Zhang, and Yongrui Qin. 2018. Collaborative location recommendation by integrating multi-dimensional contextual information. *ACM Transactions on Internet Technology (TOIT)* 18, 3 (2018), 1–24.

[32] Jianbo Yuan, Walid Shalaby, Mohammed Korayem, David Lin, Khalifeh AlJadda, and Jiebo Luo. 2016. Solving cold-start problem in large-scale recommendation engines: A deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 1901–1910.

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

[34] Liang Zhao, Yang Wang, Daxiang Dong, and Hao Tian. 2019. Learning to Recommend via Meta Parameter Partition. *arXiv preprint arXiv:1912.04108* (2019).

[35] Yu Zhu, Jinghao Lin, Shibi He, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. 2019. Addressing the item cold-start problem by attribute-driven active learning. *IEEE Transactions on Knowledge and Data Engineering* (2019).

A APPENDIX

A.1 Dataset Details

A.1.1 Movielens. The user profile includes gender (male or female), age group (under 18, 18-25, 26-35, 36-45, 46-50, 51-56, 56+), and 21 different occupations (e.g. student, engineer). The item profile includes movie's release year, genres (e.g. action or adventure), director (one or more directors), and rate (e.g. R, PG). The mean rating value is 3.58, and the rating comment time is between 2000-04-26 and 2003-03-01. In the raw data, each user has minimum 20 rating histories. We sort the items rated by a user according to their review time, and we trim the dataset, i.e. each user having 20 rating records, to force the model to learn from few shot cases.

A.1.2 Book-crossing. The raw Book-crossing data contains many missing and misleading values. The user information includes user age and location. The original user age ranges from 0 to 237, which is in contrast of real cases. Thus, we control the age interval as 5 to 110. The location includes city, state, and country. We only keep the country information for both missing value and privacy consideration. The filtered data has users from 65 different countries. The item information includes the publication year (ranges from 1500 to 2010), author (25593 unique authors), and publisher (5254 unique publishers). The Book-crossing dataset does not contain time information for the ratings, thus we assume the data stored in the public dataset are based on the review time, where we keep the order of items that rated by a user to make our dataset. We also keep 20 rating records for each user for few-shot learning.

A.2 Compared methods

A.2.1 Code. The code of MeLU[4] and S^2 Meta[5] are provided by the authors. We modify the input and evaluation modules to fit our experimental settings. For MetaCS-DNN, which has similar idea of MeLU, we modify the code of MeLU to implement it. As for RUM, the code is not published; thus, we implement them with Pytorch.

A.2.2 Parameter configuration. For MeLU, we use the default parameter settings in the published code. For S^2 Meta, we implement the code with the parameter settings for Movielens Dataset. For other two comparison methods, we use the suggested parameters when reproduce them. For MetaCS-DNN, the global update learning rate is set to 0.4. For RUM, the learning rate of SGD is determined by grid search in the range of [1, 0.1, 0.01, 0.001, 0.0001], and the number of memory slot K is empirically set as 20. The MERGE parameter α is searched in the range of [0,1] with step 0.1. The embedding dimension and regularization parameters are determined by grid search in the range of [10, 20, 30, 40, 50] and [0.1, 0.01, 0.001, 0.0001], respectively.

A.2.3 The evaluation metrics in cold-scenarios. We provided our definitions for cold-users and cold-items in section 3.1.4 and evaluated the performance under the evaluation metrics *MAE* (see Eq. 13) and *NDCG@N* (see Eq. 14). Notice that the users could be either warm users or cold users, and the items rated by a user could be either warm items or cold items – we label each rating record as in four cold-start scenarios. For example, for a cold user, the rated

[4]https://github.com/hoyeoplee/MeLU
[5]https://github.com/THUDM/ScenarioMeta

items are labeled as either warm items (C-W) or cold items (C-C). The calculation for *MAE* in four cold-start scenarios is easy, where we can simply calculate the mean value for the ratings in the four scenarios (i.e. W-W, W-C, C-W, and C-C). While for *NDCG@N*, the query set for a user may contain less than N records in different scenarios. Thus, for each scenario, we concatenate the results; separate the results into small clips; calculate *NDCG@N* for each clip; and then take the mean value of the clips as the results.

A.3 Parameter settings.

Our code is implemented with PyTorch [6] 1.4.0 in Python 3.7 and runs on a Linux server with NVIDIA TITAN X. The processed datasets will take about 2GB hard disk space. The default activation function is LeakyRelu [7]. For Movielens dataset: the dimension of the embedding is set to 100; the default setting of the number of layers is 2; the local learning rate ρ is 0.01 and the global learning rate λ is 0.05; the hyper-parameters α, β, γ and τ are set to 0.5, 0.05, 0.1, and 0.1, respectively. For Bookcrossing dataset: the dimension of the embedding is set to 50; the default setting of the number of layers is 2; the local learning rate ρ is 0.01 and the global learning rate λ is 0.01; the hyper-parameters α, β, γ and τ are set to 0.5, 0.1, 0.1, and 0.15, respectively. The random seed may affect the results – sometimes the results may show better or worse performance than our presented results. The running time for one epoch over all users is about half an hour; so, a strategy is updating the global parameters after learning from a batch of training users. During the test process, the model updates as in algorithm 2.

Algorithm 2: Testing process of MAMO

Input: Testing user set U^{test}; User profile $\{p_u | u \in U^{test}\}$;
Item profile $\{p_i | i \in (I_u^S, I_u^Q)\}$; User ratings
$\{y_{u,i} | u \in U^{test}, i \in (I_u^S, I_u^Q)\}$; Hyper-parameters α, β,
γ, τ, ρ, λ; Meta parameters ϕ_u, ϕ_i, ϕ_r, M_P, M_U, $M_{U,I}$

Output: Predicted user preference $\{y_{u,i} | u \in U^{test}, i \in I_u^Q\}$

1 **for** $u \in U^{test}$ **do**
2   Calculate bias term $b_u \leftarrow (p_u, M_U, M_P)$ by Eq. (3-4);
3   Initialize the local parameters θ_i, θ_u, θ_r by Eq. (9-10);
4   Initialize the preference memory $M_{u,I} \leftarrow (p_u, M_{U,I})$ by
     Eq. (7);
5   **for** $i \in I_u^S$ **do**
6   Get user and item embedding e_u and e_i by Eq. (1);
7   Get prediction of $\hat{y}_{u,i}$ by Eq. (2);
8   Local update $M_{u,I}$;
9   Local update θ_u, θ_i, θ_r by:
         $\theta_* \leftarrow \theta_* - \rho \cdot \nabla_{\theta_*} \mathcal{L}(y_{u,i}, \hat{y}_{u,i})$;
10   **end**
11   **for** $i \in I_u^Q$ **do**
12   Get user and item embedding e_u and e_i by Eq. (1);
13   Get prediction of $\hat{y}_{u,i}$ by Eq. (2);
14   **end**
15 **end**

---

[6]https://pytorch.org/
[7]https://en.wikipedia.org/wiki/Rectifier_(neural_networks)