How to Retrain Recommender System? A Sequential Meta-Learning Method[*]

Yang Zhang[1], Fuli Feng[2], Chenxu Wang[1], Xiangnan He[1], Meng Wang[3], Yan Li[4], Yongdong Zhang[1]
[1]University of Science and Technology of China, [2]National University of Singapore
[3]Hefei University of Technology, [4]Beijing Kuaishou Technology Co., Ltd. Beijing, China
{fulifeng93,xiangnanhe}@gmail.com,{zy2015,wcx123}@mail.ustc.edu.cn
eric.mengwang@gmail.com,liyan@kuaishou.com,zhyd73@ustc.edu.cn

ABSTRACT

Practical recommender systems need be periodically retrained to refresh the model with new interaction data. To pursue high model fidelity, it is usually desirable to retrain the model on both historical and new data, since it can account for both long-term and short-term user preference. However, a full model retraining could be very time-consuming and memory-costly, especially when the scale of historical data is large. In this work, we study the model retraining mechanism for recommender systems, a topic of high practical values but has been relatively little explored in the research community.

Our first belief is that retraining the model on historical data is unnecessary, since the model has been trained on it before. Nevertheless, normal training on new data only may easily cause overfitting and forgetting issues, since the new data is of a smaller scale and contains fewer information on long-term user preference. To address this dilemma, we propose a new training method, aiming to abandon the historical data during retraining through learning to transfer the past training experience. Specifically, we design a neural network-based transfer component, which transforms the old model to a new model that is tailored for future recommendations. To learn the transfer component well, we optimize the "future performance" — i.e., the recommendation accuracy evaluated in the next time period. Our *Sequential Meta-Learning* (SML) method offers a general training paradigm that is applicable to any differentiable model. We demonstrate SML on matrix factorization and conduct experiments on two real-world datasets. Empirical results show that SML not only achieves significant speed-up, but also outperforms the full model retraining in recommendation accuracy, validating the effectiveness of our proposals. We release our codes at: https://github.com/zyang1580/SML.

CCS CONCEPTS

• **Information systems → Recommender systems**.

[*]This work is supported by the National Natural Science Foundation of China (61972372, U19A2079,61725203). Fuli Feng is the Corresponding Author.

KEYWORDS

Recommendation; Model Retraining; Meta-Learning

ACM Reference Format:
Yang Zhang, Fuli Feng, Chenxu Wang, Xiangnan He, Meng Wang, Yan Li, Yongdong Zhang. 2020. How to Retrain Recommender System? A Sequential Meta-Learning Method[*]. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401167

1 INTRODUCTION

Recommender systems play an increasingly important role in the current Web 2.0 era which faces with serious information overload issues. The key technique in a recommender system is the personalization model, which estimates the preference of a user on items based on the historical user-item interactions [14, 33]. Since users keep interacting with the system, new interaction data is collected continuously, providing the latest evidence on user preference. Therefore, it is important to retrain the model with the new interaction data, so as to provide timely personalization and avoid being stale [36]. With the increasing complexity of recommender models, it is technically challenging to apply real-time updates on the models in an online fashion, especially for those expressive but computationally expensive deep neural networks [13, 26, 43]. As such, a common practice in industry is to perform model retraining periodically, for example, on a daily or weekly basis. Figure 1 illustrates the model retraining process.

Intuitively, the historical interactions provide more evidence on user long-term (e.g., inherent) interest and the newly collected interactions are more reflective of user short-term preference. To date, three retraining strategies are most widely adopted, depending on the data utilization:

- Fine-tuning, which updates the model based on the new interactions only [35, 41]. This way is memory and time efficient, since only new data is to be handled. However, it ignores the historical data that contains long-term preference signal, thus can easily cause overfitting and forgetting issues [6].
- Sample-based retraining, which samples historical interactions and adds them to new interactions to form the training data [6, 42]. The sampled interactions are expected to retain long-term preference signal, which need be carefully selected to obtain representative interactions. In terms of recommendation accuracy, it is usually worse than using all historical interactions due to the information loss caused by sampling [42].
- Full retraining, which trains the model on the whole data that includes all historical and new interactions. Undoubtedly, this

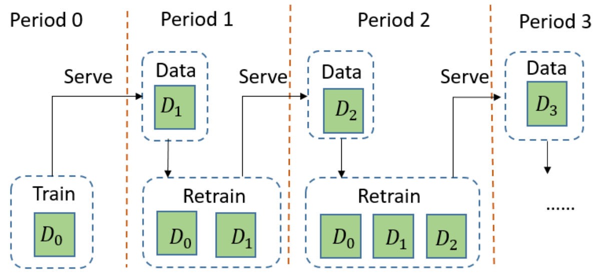


Figure 1: An illustration of periodical model retraining.

method costs most resources and training time, but it provides the highest model fidelity since all available interactions are utilized.

While the above three strategies have their pros and cons, we argue a key limitation is that they lack an explicit optimization towards the retraining objective — i.e., the retrained model should serve well for the recommendations of the next time period. In practice, user interactions of the next time period provide the most important evidence on the generalization performance of the current model, and are usually used for model selection or validation. As such, an effective retraining method should take this objective into account and formulate the retraining process towards optimizing the objective, a much more principled way than manually crafting heuristics to select data examples [6, 35, 40, 42].

In this work, we explore the central theme of model retraining in recommendation, a topic of high practical value in industry recommender systems but receives relatively little scrutiny in research. Although full model retraining provides the highest fidelity, we argue that it is not necessary to do so. The key reason is that the historical interactions have been trained in the previous training, which means the model has already distilled the "knowledge" from the historical data. If there is a way to retain the knowledge well and transfer it to the training on new interactions, we should be able to keep the same performance level as the full retraining, even though we do not use the historical data during model retraining. Furthermore, if the knowledge transfer is "smart" enough to capture more patterns like recent data is more reflective of near future performance, we even have the opportunity to improve over the full retraining in recommendation accuracy.

To this end, we propose a new retraining method with two major considerations: (1) building an expressive component that transfers the knowledge gained in previous training to the training on new interactions, and (2) optimizing the transfer component towards the recommendation performance in the near future. To achieve the first goal, we devise the transfer component as a convolutional neural network (CNN), which inputs the previous model parameters as constant and the present model as trainable parameters. The rationality is that the knowledge gained in previous training is condensed in model parameters, such that an expressive neural network should be able to distill the knowledge towards the desired purpose. To achieve the second goal, in addition to normal training on newly collected interactions, we further train the transfer CNN on the future interactions of next time period. As such, the CNN can learn how to combine the old parameters with present parameters, with the objective of predicting the user interactions of the near future. The whole architecture can be seen as an instance of meta-learning [9]: the retraining of each time period is a task, which has the new interactions of the current period as the training set and the future interactions of the next period as the testing set. By learning to train historical tasks well, we expect the method to perform well for future tasks. Since our meta-learning mechanism is operated on sequential data, we name it as *Sequential Meta-Learning* (SML).

The main contributions of this work are summarized as follows:

- We highlight the importance of recommender retraining research and formulate the sequential retraining process as an optimizable problem.
- We propose a new retraining approach that is 1) efficient by training on new interactions only, and 2) effective by optimizing for the future recommendation performance.
- We conduct experiments on two real-world datasets of Adressa news and Yelp business. Extensive results demonstrate the effectiveness and rationality of our method.

2 PROBLEM FORMULATION

In real-world recommender systems, user interaction data streams in continuously. To keep the predictive model fresh with recent data, a common choice is to retrain the model periodically. We represent the data as $\{D_0, \ldots, D_t, D_{t+1}, \ldots\}$, where D_t denotes the data newly collected in the time period t. Assume each retraining is triggered right after D_t is collected. A period can be any length of time, e.g., daily, weekly or until a certrain number of interactions are collected, depending on the system requirement and implementation abilty.

In the retraining of time period t, the system has access to all previous data, i.e., $\{D_0, \ldots, D_{t-1}\}$, and the new data D_t. Since the retrained model is used to serve for the near future, it is reasonable to judge its effectiveness based on D_{t+1} — the data collected in the next time period. As such, we set the recommendation performance on D_{t+1} as the generalization goal of the t-th period retraining. Let the model parameters after the t-th peirod retraining be W_t. We treat each retraining as a *task*, formulating it as:

$$(\{D_m : m \leq t\}, W_{t-1}) \xrightarrow{get} W_t \xleftarrow{test} D_{t+1}. \tag{1}$$

That is, based on all accessible data at the time of retraining and the model parameters of the previous retraining, we aim to get a new set of model parameters that can perform well on the near future data D_{t+1}. The mostly used solution in industry is to perform a full retraining on the whole data with W_{t-1} as the initialization. This solution is straightforward to implement. However, the drawback is that it takes too many computation resources, a relative long retraining time, and requires to enlarge the computation power as time goes by. Another limitation is that the full retraining lacks explicit optimization for the performance on D_{t+1}. This is non-trivial to address, since directly using D_{t+1} in training will cause information leak and worse generalization ability.

In this work, we aim to utilize the newly collected data D_t only plus the previous model parameters W_{t-1}, so as to pursue a good retrained model as evaluated on D_{t+1}. Thus we reformulate the retraining process as:

$$(D_t, W_{t-1}) \xrightarrow{get} W_t \xleftarrow{test} D_{t+1}, \tag{2}$$

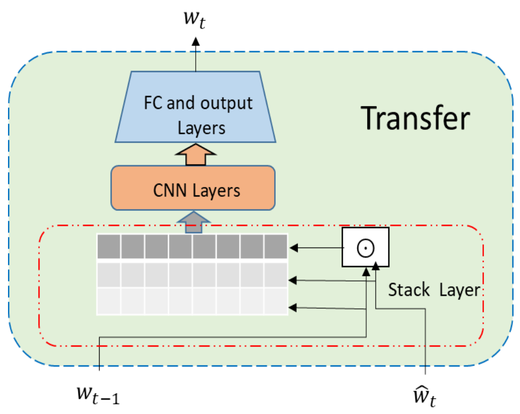


Figure 2: Model overview of our transfer-based retraining for the t-th time period. W_{t-1} represents the previous recommender, $\hat{W}_t$ is a recommender learned on new data D_t only. The transfer component is to combine the "knowledge" in W_{t-1} and $\hat{W}_t$ to obtain the new recommender W_t for serving the next period.

which we denote as the task τ_t. For τ_0, the previous model parameters are just random initialization. A straightforward solution is to perform stochastic gradient descent (SGD) updates on D_t with W_{t-1} as initialization. However, it is easy to encounter the forgetting issue of user long-term interest, since the effect of initialization is weakening with more updates. Moreover, this solution also lacks optimization scheme towards serving D_{t+1}.

Distinct from the definition of *task* in standard meta-learning [9, 21], the tasks here naturally form a sequence $\{\tau_0, ..., \tau_t, \tau_{t+1}, ...\}$. In online serving (testing), only if τ_t has been completed we can move to τ_{t+1}. As such, the offline training should follow the similar manner of sequential training to ensure the method can generalize well in future serving. Lastly, addressing the problem can be seen as an instance of meta-learning, since the learning target is how to solve the tasks well (i.e., with a good generalization ability on future tasks), which is a higher-level problem than simply learning model parameters on D_t.

3 METHOD

Firstly, we present the model overview to solve the task τ_t, the core of which is to design a transfer component that effectively converts the old model W_{t-1} to a new model W_t. Then, we elaborate our design of the transfer. Next, we discuss how to train the model with good performance on current data D_t as well as good generalization to future data D_{t+1}. Lastly, we demonstrate how to instantiate our generic method on matrix factorization, one of the most classic and representative models for collaborative filtering.

3.1 Model Overview

We aim to solve the task τ_t defined in Equation (2) which leverages only the new data D_t to achieve a comparable or even better performance than the full retraining. The belief is that the past data $\{D_0, ..., D_{t-1}\}$ have been seen in previous training, such that the "knowledge" useful for recommendation has been gained and stored in model parameters W_{t-1}. Another consideration is to make our method technically applicable to many recommender models, rather than a specific one.

To this end, we design a generic model framework, as illustrated in Figure 2. It has three components: 1) W_{t-1} represents the previous recommender model that is trained from past data, 2) $\hat{W}_t$ denotes a new recommender model that needs to be learned from the current data D_t, and 3) Transfer is the module to combine the "knowledge" contained in W_{t-1} and $\hat{W}_t$ to form a new recommender model W_t, which is used for serving next period recommendations. In the t-th period retraining, W_{t-1} is set as constant input, and the retraining consists of two main steps:

1. Obtaining $\hat{W}_t$, which is expected to contain useful signal for recommendation from D_t. This step can be done by optimizing standard recommendation loss, denoted as $L_r(\hat{W}_t|D_t)$.
2. Obtaining W_t, which is the output of the transfer module:
$$W_t = f_\Theta(W_{t-1}, \hat{W}_t) \tag{3}$$
where f_Θ denotes the transfer function, Θ denotes its parameters, and W_{t-1} and $\hat{W}_t$ are its input.

In this framework, W_{t-1} and $\hat{W}_t$ can be any differentiable recommender model, as long as they are of the same architecture (i.e., the parameter number and semantics are the same). Only the transfer component needs to be carefully designed, which is our contribution to be introduced next.

3.2 Transfer Design

Functionally speaking, the transfer combines parameters W_{t-1} and $\hat{W}_t$ to form a new group of parameters W_t. As the most basic requirement, W_t needs be of the same shape with W_{t-1} and $\hat{W}_t$. This requirement can be easily satisfied by operations like weighted sum:
$$W_t = \alpha W_{t-1} + (1 - \alpha)\hat{W}_t,$$
where α is the combination coefficient which can be either pre-defined or learned. The method is simple to interpret by paying different attentions to previous and current trained knowledge; it is also easy to train, since few parameters are introduced. However it has limited representation ability, for example, cannot account for the relations between different dimensions of parameters.

For expressiveness of the transfer, multi-layer perceptron (MLP) can be another option:
$$W_t = MLP(W_{t-1}||\hat{W}_t).$$
Despite the universal approximation theorem of MLP [19], it may be practically difficult to be trained well [1, 13]. Another limitation is that it does not emphasize the interactions beweeen the parameters of the same dimension, which could be important for understanding parameter evolution. As an example, suppose the model is matrix factorization and the parameters are user embedding. Then the difference $\hat{W}_t - W_{t-1}$ means parameter change which can capture the interest drift; and each dimension of the product $W_{t-1} \odot \hat{W}_t$ indicates the importance of the dimension in reflecting user interest of both short-term and long-term. However, MLP lacks mechanisms to explicitly capture such patterns.

To this end, we design the transfer component to be capable of not only emphasizing the relation between W_{t-1} and $\hat{W}_t$ at each dimension, but also capturing the relations among different dimensions. Inspired by the success of convolutional neural network (CNN) in capturing local-region features in image processing, we design the transfer based on CNN. The CNN architecture can be found in the green box of Figure 2, which consists of a stack layer, two convolution layers, and a fully connected layer for output.

Next we detail the CNN design. Without loss of generality, we treat W_{t-1} and $\hat{W}_t$ as a row vector, denoted as w_{t-1} and $\hat{w}_t$, respectively, even though their original form can be matrix or tensor. This facilitates us performing dimension-wise operations on combining two models.

Stack layer. This layer stacks w_{t-1}, $\hat{w}_t$, and their element-wise product interaction vector as a 2D matrix, which serves as an "image" to be processed by the later convolution layers. Specifically, we formulate it as:

$$H^0 = \begin{bmatrix} w_{t-1} \\ \hat{w}_t \\ w_{dot} \end{bmatrix}, \text{where } w_{dot} = \frac{w_{t-1} \odot \hat{w}_t}{\|w_{t-1}\| + \epsilon}. \tag{4}$$

The $w_{t-1} \odot \hat{w}_t$ can capture that when w_{t-1} evolves to $\hat{w}_t$, which dimension values are enlarged or diminished. The denominator of w_{dot} is used for normalization, and $\epsilon = 10^{-15}$ is a small number to prevent the denominator being zero. The size of H^0 is $3 \times d$, where d denotes the size of w_{t-1} and $\hat{w}_t$.

Convolution layers. H^0 is fed into two cascaded convolution layers that further model dimension-wise relations. We describe the first convolution layer since the second one is formulated similarly. Let the first convolution layer have n_1 vertical filters, where each filter is denoted as $F_j \in \mathbb{R}^{3\times1}$ (where $j = 1, ..., n_1$ denotes the filter index). F_j slides from the first column to the last column of H^0 to perform operations on each column vector:

$$H^1_{j,m} = \text{GELU}(< F_j\ ,\ H^0_{:,m} >), \tag{5}$$

where $H^0_{:,m}$ is the m-th column vector of H^0, $<, >$ denotes vector inner product, and $H^1_{j,m} \in \mathbb{R}$ is the convolution result of F_j on $H^0_{:,m}$. GELU is the Gaussian Error Linear Units activation function [17], which can be seen as a smoothed variant of ReLU with gradients for negative values.

Note that the vertical filter F_j can learn various relations between $\hat{w}_t$ and w_{t-1} at the same dimension. For example, if the filter is $[-1, 1, 0]$, it can express the difference between $\hat{w}_t$ and w_{t-1}; if the filter is $[1, 1, 1]$, it can obtain prominent features that have high positive value on both $\hat{w}_t$ and w_{t-1}. Another reason we use such 1D filter rather than the standard 2D filters is that the dimension order in w_{t-1} or $\hat{w}$ is not meaningful for many recommender models. For example, if we permutate the embedding order for factorization models, the model prediction will not be changed.

The output of the first convolution layer H^1 is a matrix of size $n_1 \times d$, which is then fed into the second convolution layer of n_2 filters, where each filter is of size $n_1 \times 1$. As a result, we obtain the output of this component H^2, which is a matrix of size $n_2 \times d$.

Full-connected and output layers. H^2 is fed into a fully-connected (FC) layer to capture the relations among different dimensions. We first flatten H^2 as a vector which has the size dn_2, and then feed into a fully connected layer:

$$z = \text{GELU}(W_f^T \text{flatten}(H^2) + b_1), \tag{6}$$

where $W_f \in \mathbb{R}^{(dn_2)\times d_f}$ and $b_1 \in \mathbb{R}^{d_f}$ are the weight matrix and bias vector of the FC layer, respectively, and d_f denotes the layer size. The vector $z \in \mathbb{R}^{d_f}$ is then transformed by a linear layer to output the new parameter vector w_t:

$$w_t = W_o^T z + b_2, \tag{7}$$

where $W_o \in \mathbb{R}^{d_f \times d}$ and $b_2 \in \mathbb{R}^d$ are the weight matrix and bias vector of the linear layer, respectively. Lastly, the parameter vector w_t is reshaped to the W_t — i.e., the new model parameters after retraining.

To summarize, all trainable parameters of the transfer component are $\Theta = \{F^{(1)}, F^{(2)}, W_f, b_1, W_o, b_2\}$, where $F^{(1)} \in \mathbb{R}^{n_1\times3}$ and $F^{(2)} \in \mathbb{R}^{n_2\times n_1}$ denote the filters of the first and second convolution layer, respectively. It is worth mentioning that we can categorize the parameters of a recommender model into different groups, and apply a separate transfer network for each group. For example, the matrix factorization model has two groups of parameters — user embedding and item embedding. Then we use two transfer networks, one for user embedding and another for item embedding (see details in Section 3.4).

3.3 Sequential Training

We now consider how to train model parameters, including the transfer input $\hat{W}_t$ for each task τ_t, and the tranfer parameter Θ that is shared for all tasks. Functionally speaking, $\hat{W}_t$ is expected to extract recommendation knowledge from the current data D_t, whereas Θ combines the previous model W_{t-1} and $\hat{W}_t$, which is expected to make the transfer output W_t perform well on future data D_{t+1}. Since the data comes in sequentially, we perform training in the same sequential way, i.e., solving the task τ_{t-1} before moving to the next task τ_t. Algorithm 1 shows the sequential training process. We next describe how to train for a task τ_t (i.e., line 3 to 11), which has two main steps:

Step 1. Learning the transfer input $\hat{W}_t$. A straightforward solution is to directly learn it based on the recommendation loss on D_t. However, the resultant $\hat{W}_t$ may not be suitable as the input to the transfer, which assumes W_{t-1}, $\hat{W}_t$, and W_t are in the same space (i.e., the parameter dimensions are aligned for the same semantics and the values are in the same scale range). To address this problem, we propose to optimize the transfer output on D_t, back-propogating gradients to the transfer input $\hat{W}_t$. Specifically, we formulate the loss as:

$$L_r(\hat{W}_t|D_t) = L_0(f_\Theta(W_{t-1}, \hat{W}_t)|D_t) + \lambda_1||\hat{W}_t||^2, \tag{8}$$

where $L_0(x|D_t)$ denotes the recommendation loss (e.g., the log loss [15] or pairwise loss [33]) on data D_t with x as the recommender model parameters (note $x = f_\Theta(W_{t-1}, \hat{W}_t)$ here). λ_1 is a hyper-parameter to control L_2 regularization to prevent overfitting. When optimizing the loss, Θ is treated as constant and is not updated, so only the gradient of $\hat{W}_t$ needs be evaluated:

$$\frac{\partial L_r(\hat{W}_t|D_t)}{\partial \hat{W}_t} = \frac{\partial L_0(x|D_t)}{\partial x} \cdot \frac{\partial f_\Theta(W_{t-1}, \hat{W}_t)}{\partial \hat{W}_t} + 2\lambda_1\hat{W}_t, \tag{9}$$

After getting this gradient, we can apply gradient descent optimizer to update $\hat{W}_t$ like SGD and Adam [22]. Through this way, we can achieve the two effects simultaneously 1) distilling recommendation knowledge from D_t, and 2) making $\hat{W}_t$ suitable as the input to the transfer network.

Step 2. Learning the transfer parameter Θ. Since Θ is shared across all tasks, it can capture some task-invariant patterns, e.g., which parameter dimensions are more relective of user short-term interests and should be emphasized when combining W_{t-1} and $\hat{W}_t$. The general aim is to obtain such patterns that are tailored for the next-period recommendations. As such, we consider optimizing Θ on the next-period data D_{t+1}. Specifically, we formulate the objective function as:

$$L_s(\Theta|D_{t+1}) = L_0(f_\Theta(W_{t-1}, \hat{W}_t)|D_{t+1}) + \lambda_2||\Theta||^2, \tag{10}$$

where λ_2 is regularization hyper-parameter. Note that the $\hat{W}_t$ gained in Step 1 is a function of Θ. Thus, when computing the gradients of Θ, it will cause high-order gradients that are expensive to obtain. As such, we follow the first-order MAML algorithm [9], ignoring such high-order gradients which have minor impacts on gradients but are expensive to obtain. By treating $\hat{W}_t$ as constant in this step, we evaluate the gradient of Θ as:

$$\frac{\partial L_s(\Theta|D_{t+1})}{\partial \Theta} = \frac{\partial L_0(x|D_{t+1})}{\partial x} \cdot \frac{\partial f_\Theta(W_{t-1}, \hat{W}_t)}{\partial \Theta} + 2\lambda_2\Theta, \tag{11}$$

where $x = f_\Theta(W_{t-1}, \hat{W}_t)$ for brevity.

The above two steps are iterated until convergence or a maximum number of iterations is reached (line 4). As Algorithm 1 shows, the update of Θ is not performed in the last training period T, since its next period data D_{T+1} is not available in training. Note that we can run multiple passes of such sequential training on $\{D_t\}_{t=0}^T$, while we empirically find one pass is sufficient to obtain good performance, thus we train only one pass.

It is worth mentioning that the parameter update procedure of the serving (evaluation) phase slightly differs. Algorithm 2 shows how we perform model evaluation for testing (validation) with newly collected data D_{t+1}. First, we use it to test the model W_t that served the period $t+1$. Then, we need to update Θ and $\hat{W}_{t+1}$ with D_{t+1}, so as to obtain W_{t+1} for serving next period (note that $W_{t+1} = f_\Theta(W_t, \hat{W}_{t+1})$). As shown in line 3-8, we first iterate the updating of Θ and $\hat{W}_t$, which is same as the training phase. When stopping condition meets, we used the refreshed Θ to update $\hat{W}_{t+1}$, which is finally fed into $f_\Theta(W_t, \hat{W}_{t+1})$ to achieve W_{t+1}.

```
Algorithm 1: Sequential Training of SML
   Input: Training data of $T$ periods $\{D_t\}_{t=0}^T$
   Output: Recommender $W_T$, transfer $\Theta$
1  Randomly initialize $W_{-1}$ and $\Theta$ ;
2  for $t = 0$ to $T$ do
3      $\hat{W}_t \leftarrow W_{t-1}$ ;
4      while Stop condition is not reached do
5          // Step 1: Learning $\hat{W}_t$
6          Update $\hat{W}_t$ by optimizing $L_r(\hat{W}_t|D_t)$;
7          // Step 2: Learning $\Theta$
8          if $t == T$ then break ;
9          Update $\Theta$ by optimizing $L_s(\Theta|D_{t+1})$;
10     end
11     $W_t \leftarrow f_\Theta(W_{t-1}, \hat{W}_t)$ ;
12 end
13 return $W_T, \Theta$
```

```
Algorithm 2: Model evaluation and update
   Input: Newly collected data $D_{t+1}$, recommender $W_t$ to test
   Output: Updated recommender $W_{t+1}$
1  Use $D_{t+1}$ to test the model $W_t$;
2  // Model update for next period;
3  while Stop condition is not reached do
4      Update $\Theta$ by optimizing $L_s(\Theta|D_{t+1})$ ;
5      Update $\hat{W}_t$ by optimizing $L_r(\hat{W}_t|D_t)$ ;
6  end
7  Run line 4 and $W_t \leftarrow f_\Theta(W_{t-1}, \hat{W}_t)$ ;
8  Update $\hat{W}_{t+1}$ by optimizing $L_r(\hat{W}_{t+1}|D_{t+1})$ ;
9  $W_{t+1} = f_\Theta(W_t, \hat{W}_{t+1})$ ;
10 return $W_{t+1}$
```

3.4 Instantiation on Matrix Factorization

To demonstrate how our proposed SML framework works, we provide an implementation based on matrix factorization (MF), a representative embedding model for recommendation. Given a user-item pair (u, i), MF predicts the interaction score as:

$$\hat{y}_{ui} = p_u^T q_i, \tag{12}$$

where $p_u \in \mathbb{R}^{dim}$ and $q_i \in \mathbb{R}^{dim}$ denote the embedding of user u and item i, respectively, and dim denotes the embedding size. As we can see, MF has two groups of parameters: user embedding and item embedding. Thus we build two separate transfer networks, one for user embedding and another for item embedding. Instead of feeding the embeddings of all users into the user transfer network, we operate the transfer network on the basis of each user embedding; same for the item side. The rationality is that the semantics of embedding dimensions across all users are the same, thus we can share the transfer network for all users. This largely reduces the number of transfer parameters and makes the transfer more generalizable.

For the recommendation loss L_0, we adopt the pointwise log loss, which is a common choice for recommender training [15, 26]. For each interaction $(u, i) \in D_t$, we randomly sample 1 unobserved interactions of u to form the negative data set D_t^-. Then the log loss is formulated as:

$$L_0(P, Q|D_t) = -\sum_{(u,i)\in D_t} \log(\sigma(\hat{y}_{ui})) - \sum_{(u,j)\in D_t^-} \log(1 - \sigma(\hat{y}_{uj})), \tag{13}$$

where P and Q denote the embeddings of all users and items, $\sigma(\cdot)$ denotes the sigmoid function. The same log loss is used in both optimization of the recommender L_r and the transfer L_s.

4 EXPERIMENTS

We conduct experiments to answer the following questions:

RQ1: How is the performance of SML compared with existing retraining strategies and recommender models?
RQ2: How do the components of SML affect its effectiveness?
RQ3: How does the CNN architecture affect the transfer network?
RQ4: Where are the improvements of SML come from?

We first present experimental settings, followed by results and analyses to answer each research question.

4.1 Experimental Settings

4.1.1 Datasets. We experiment with two real-world datasets from Adressa and Yelp.

Yelp: The dataset is adopted in Yelp Challenge 2019[1], which contains the interaction records between users and businesses like restaurants and bars, spanning a period of more than 10 years. For ease of evaluation, we remove the inactive users with less than 10 interactions and unpopular items with less than 20 interactions. The experimented data contains 3,014,421 interactions from 59,082 users and 122,816 items.

Adressa [11]: The dataset is from Adressa[2], which records user clicks on news articles in three weeks. We remove invalid interactions that have a news reading time of zero. The experimented data has 3,664,225 interactions between 478,612 users and 20,875 items.

We purposefully choose the two datasets because of their different properties — the Adressa dataset emphasizes more on user short-term interest, since the news domain is more time-sensitive; in contrast, the Yelp dataset emphasizes more on user long-term interest, since it lasts longer and a user's choice on businesses is less time-sensitive.

To test the periodical model retraining, we organize each dataset into periods (*i.e.,* $\{D_0, \ldots, D_T\}$) according to the timestamp of interaction. For Adressa, we split each day into three periods based on the morning (0:00-10:00), afternoon (10:00-17:00), and evening (17:00-24:00), obtaining 63 periods in total. As for the Yelp dataset which lasts longer, we split it into 40 periods with an equal number of interactions, where each period roughly corresponds to a quarter. We further split the periods of each dataset into training/validation/testing sets: for Adressa the ratio is 48/5/10, and for Yelp the ratio is 30/3/7. For each testing period, data collected in all the previous periods can be used for model retraining.

4.1.2 Baselines. We compare the proposed SML method with three *retraining strategies* that are also applied to MF:

- Full-retrain. This method trains the MF model on all past data $\{D_0, ..., D_{t-1}\}$ and newly collected data D_t at each period t.

- Fine-tune. This method updates the MF model on the newly collected data D_t only.

- SPMF [42]. This is a state-of-the-art streaming recommendation method that belongs to the category of sampled-based retraining. It maintains a reservoir of historical interaction samples and adds them into D_t to retrain the MF model. We tune the reservoir size in {7000,15000,30000,70000}.

We also compare with two *sequential recommendation* methods, which are designed for modeling sequential user-item interactions:

- GRU4Rec [18]. This is a representative sequential recommender based on recurrent neural network (RNN). It builds a RNN for each user's interaction sequence to capture her interest evolution. We employ full retraining strategy at each testing period, which performs better than fine-tuning for the method, and keep loss as the paper. The hidden layer size of GRU is tuned in the range of $\{64, 128, 256\}$.

- Caser [38]. This method uses CNN for sequential modeling. It takes L most recently interacted items and forms their embeddings as a 2D matrix, feeding the matrix into a CNN with two types of convolution layers — horizontal layer and vertical layer. We tune L in the range of $\{2, ..., 5\}$,CNN kernel number in $\{4, 8, 16\}$, and other hyper-parameters follow the optimal setting as reported in the paper. We employ fine-tuning strategy at each testing period, which performs better than full training for the method.

For fair comparison, all methods are optimized with the same log loss (except GRU4Rec) and tuned on the validation set. For the four methods based on MF (*i.e.,* Full-retrain, Fine-tune, SPMF, and our SML), we tune three hyper-parameters: L_2 regularization coefficient λ in $\{1e\text{-}1, 1e\text{-}2, ..., 1e\text{-}7, 0\}$, learning rate in $\{0.1, 0.01, 0.001\}$, and training epochs in {5,10,20,50,100}, respectively. For SML, we additionally tune the number of maximum iterations (line 4 of Algorithm 1) in $\{5, 6, \ldots, 10\}$. The CNN filter size is set as $[10, 5]$ and the fully connected layer size is 512 for both datasets.

4.1.3 Evaluation Protocols. To simulate the real-world scenario that there are typically some historical data to train an initial model, we start model retraining from the 10-th and 20-th period of Yelp and Adressa, respectively, using the previous data to train an initial model. We perform evaluation at each testing period and report the average scores. The evaluation is done on each interaction basis. As it is time consuming to rank all non-interacted items, we sample 999 non-interacted items of a user as the recommendation candidates. For each testing interaction, the method outputs a ranking list on the 1 interacted item and 999 non-interacted items. We adopt two widely-used evaluation metrics: Recall@K and NDCG@K [15] and K is set to 5,10, and 20. For parameters tuning on validation sets, we take Recall@20 as the main referential metric.

4.2 Performance Comparison (RQ1)

4.2.1 Overall Comparison. Table 1 shows the top-K recommendation performance of compared methods. From the table, we have the following observations:

- Full-retrain outperforms Fine-tune on Yelp, but it is significantly worse on Adressa. This shows the varying properties of the two datasets: Yelp users choose businesses based more on their inherent (long-term) interest, whereas Adressa users are more time-sensitive to choose recent news and driven by their short-term interest. Full-retrain makes use of all data to do model retraining, which is effective in capturing user long-term interest; however, it suffers from emphasizing the importance of recent data. This demonstrates the necessity of properly handling both long-term and short-term user preference in the periodical model retraining.
- Our proposed SML achieves the best performance on both datasets, consistently outperforming Full-retrain and the most competitive baseline Caser. This result signifies the effectiveness

[1]https://www.yelp.com/dataset/
[2]http://reclab.idi.ntnu.no/dataset/

Table 1: Average recommendation performance over online testing periods on Adressa and Yelp. "RI" indicates the relative improvement of SML over the corresponding baseline.

Datasets	Methods	recall@5	recall@10	recall@20	RI	NDCG@5	NDCG@10	NDCG@20	RI
Adressa	Full-retrain	0.0495	0.0915	0.1631	319.7%	0.0303	0.0437	0.0616	393.1%
	Fine-tune	0.1085	0.2235	0.3776	82.8%	0.0594	0.0962	0.1351	135.5%
	SPMF	0.1047	0.2183	0.3647	87.3%	0.0572	0.0935	0.1306	143.6%
	GRU4Rec	0.0213	0.0430	0.0860	809.0%	0.0125	0.0194	0.0302	1018.4%
	Caser	0.2658	0.3516	0.4259	6.5%	0.1817	0.2096	0.2285	2.1%
	SML	**0.2815**	**0.3794**	**0.4498**	-	**0.1838**	**0.2156**	**0.2336**	-
Yelp	Full-retrain	0.1849	0.2876	0.4139	18.0%	0.1178	0.1514	0.1829	22.7%
	Fine-tune	0.1507	0.2386	0.3534	41.7%	0.0963	0.1246	0.1535	48.5%
	SPMF	0.1664	0.2591	0.3749	30.7%	0.1072	0.1370	0.1662	35.1%
	GRU4Rec	0.1706	0.2764	0.4158	22.8%	0.1080	0.1420	0.1771	30.5%
	Caser	0.2195	0.3320	0.4565	2.8%	0.1440	0.1802	0.2117	3.12%
	SML	**0.2251**	**0.3380**	**0.4748**	-	**0.1485**	**0.1849**	**0.2194**	-

of SML, which is attributed to the dedicated design of the transfer network and the sequential training algorithm. The strong performance on both datasets shows that SML is capable of adapting long-term and short-term interests by optimizing the transfer network on the next-period data.

- In particular, SML outperforms Full-retrain by 18% on Yelp. It validates our belief that historical data can be discarded during retraining, as long as the previous model can be properly utilized. This can largely save computation resources in model retraining, which has high value for practical use.
- The sample-based retraining method SPMF performs better than Fine-tune on Yelp, but not on Adressa. The reservoir in SPMF is designed heuristically to bias towards retaining old interactions, which shows strength in capturing long-term user interest. However, it sacrifices the ability of modeling short-term interest, making it fall short on recommendation scenarios where recent data are more important. Generally speaking, such sample-based retraining method pursues a trade-off between Fine-tune and Full-retrain, and its performance is bounded by either method (on Adressa SPMF is weaker than Fine-tune and on Yelp SPMF is weaker than Full-retrain).
- Among the two sequential recommender models, Caser performs much better than GRU4Rec, which implies that CNN would be a better choice than RNN to model the interaction sequence. Moreover, Caser outperforms the three MF-based baselines, which indicates its effectiveness in sequence modeling. However, its advantages can be surpassed by our SML, which wisely retrains MF towards the next-period performance. As a future extension, we will implement SML on Caser to see whether combing their advantages can lead to further improvements.

4.2.2 Period-wise Performance. Figure 3 shows the detailed recommendation performance at each online testing period evaluated by recall@10. To save space, we omit the results of other metrics which show the same trend[3]. From the figure, we can see that SML achieves the best performance in most cases, which further validates its strong generalization ability. Moreover, the fluctuations on Adressa are larger than Yelp, which further validate the strong timeliness of the news domain (*i.e.,* user interest changes quickly) and the importance of performing fast model retraining so that the recommender is adapted to the changes of short-term interest.

[3]Since GRU4Rec achieves the lowest scores, its results are not shown in the figure for better visualization.

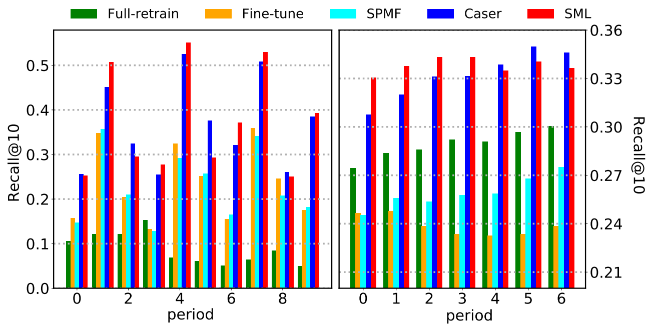


Figure 3: Recall@10 of each testing period.

Table 2: Retraining time (seconds) at each testing period on Yelp. SML-S is the variant that disables transfer update.

period	0	1	2	3	4	5	6
Full-retrain	1,458	1,492	1,546	1,599	1,634	1,701	1,749
SML	90	91	89	89	89	89	90
Fine-tune	34	34	35	34	34	35	34
SML-S	8	8	8	8	8	8	8

Full-retrain is trained with 20 epochs with recommendation performance slightly worse than that reported in Table 1 which is trained with 100 epochs.

4.2.3 Speed-up. Recall that one motivation of the work is to accelerate model retraining by avoiding using previous data. We compare the retraining time of SML with Full-retrain and Fine-tune at different testing periods. The testing platform is a 1080Ti GPU with 2 CPUs and 16GB memory. Table 2 shows the time cost on Yelp by period. As can be seen: 1) the time cost of Full-retrain increases linearly as the testing process goes on, which is caused by the increase of training data. 2) SML is about 18 times faster than Full-retrain, and the retraining time is stable across different periods. 3) By disabling the update of the transfer network in the testing process, SML-S is even faster than Fine-tune, and also outperforms Fine-tune in recommendation accuracy (see Figure 4). This shows the potential of SML in supporting fast model retraining, which is highly valuable in practice.

4.3 Ablation Studies (RQ2)

The strengths of SML come from two novel components: 1) the transfer network that combines the "knowledge" contained in the old model and new model; and 2) the sequential training process that optimizes the transfer network towards next-period performance. To justify the designs in SML, we investigate the influence of each important design. We study the performance of the following five variants:

- **SML-CNN**, it removes the CNN layers from the transfer network.
- **SML-FC**, which removes the FC layer from the transfer network.
- **SML-N**, which disables the optimization of the transfer network towards the next-period performance. It trains all parameters on D_t, i.e., replacing $L_s(\Theta|D_{t+1})$ with $L_s(\Theta|D_t)$ in Line 9 of Algorithm 1.
- **SML-S**, which disables the update of the transfer network during testing, i.e., removing Line 4 of Algorithm 2. The transfer network is fixed when updating the recommender.
- **SML-FP**, which learns the transfer input $\hat{W}_t$ directly based on the recommendation loss on D_t, rather than forward propagation through the transfer network.

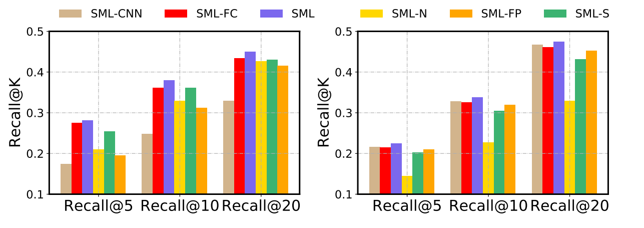
Figure 4 shows the recommendation performance of SML and the five variants on Adressa and Yelp. We omit the results of NDCG@K which show the same trend. We have the following observations:

- Regarding the design of transfer model, SML performs better than SML-CNN and SML-FC, which signifies the effectiveness of the hybrid structure with both CNN and fully connected layers. The improvement is attributed to the consideration of both dimension-wise relations and cross-dimension relations between the previous model W_{t-1} and the new model $\hat{W}_t$. This finding is consistent with prior work [7], which also verify the efficacy of jointly considering dimension-wise and cross-dimension relations in recommendation.
- Regarding the sequential training process, the performance of SML-N is worse than SML by 18.81% and 34.53% on average, which validate the advantages of optimizing towards future performance. Existing study on meta-learning [4, 9] also demonstrated the effectiveness of optimizing model parameters towards testing (validation) data. As such, it is promising to solve periodical model retraining as a sequential meta-learning task.
- When the transfer network is not updated during testing, the performance of SML (i.e., SML-S) drops by 7.87% and 9.43%. This might be caused by the drift of user interests. The performance drop signifies the importance of model retraining, which further suggests a future direction to explore online recommender updates. Lastly, SML-FP fails to achieve a comparable performance as SML on both datasets, which justifies our design of learning new model $\hat{W}_t$ based on the transfer output.

4.4 Hyper-parameter Studies (RQ3)

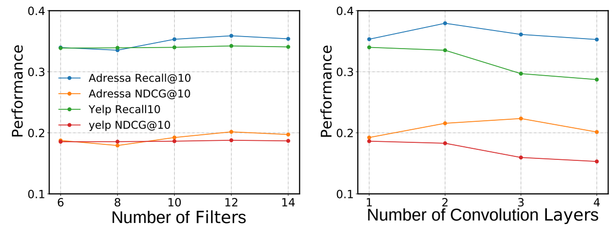
We study how the CNN architecture affects the performance, more specifically, the number of filters and the number of CNN layers.

4.4.1 Number of CNN Filters. We fix the number of convolution layer to be one and adjust the number of filters from [6,8,10,12,14]. As shown in Figure 5(a), the performance on Yelp is rather stable across different numbers of filters. While on Adressa, SML with more than 10 filters performs better than the filter number of 6 and 8. These results suggest that for applications like Adressa where the timeliness is strong, there may exist complex relations between user short-term and long-term interests. In this case, using a large number of filters is beneficial.



(a) Adressa (b) Yelp

Figure 4: Recommendation performance of SML and its five variants with different designs being disabled.



(a) Impact of filter number (b) Impact of convolution layers

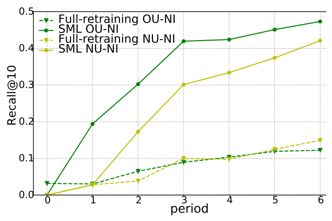
Figure 5: Performance of SML *w.r.t.* different numbers of CNN filters and CNN layers in the transfer component.

4.4.2 Number of Convolution Layers. We fix the first convolution layer with 10 filters and test the effect of stacking more convolution layers with 5 filters. As shown in Figure 5, SML achieves the best performance on Yelp with 1 convolution layer, and stacking more convolution layers degrades the performance because of overfitting. For Adressa, the best performance is achieved when the number of convolution layers is 2 or 3, and further increasing it also degrades the performance. These results suggest that the optimal number of convolution layers varies, depending on the features of the dataset.

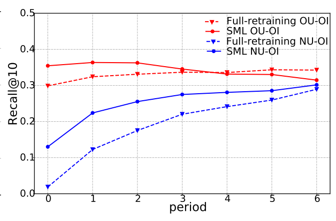
4.5 In-depth Analyses (RQ4)

We conduct in-depth analyses to understand where the improvements come from compared with the Full-retrain, and scrutinize the CNN filters to interpret their rationality.

4.5.1 Performance of Different Interaction Types. We divide users into two groups: *new users* mean the users that only occur in the testing data, otherwise *old users*; same for the item side. We then cross user groups and item groups, dividing the interaction into four types: *old user-new item* (OU-NI), *new user-new item* (NU-NI), *old user-old item* (OU-OI), and *new user-old item* (NU-OI). We then perform evaluation on each type of interactions. Figure 6 shows the performance of SML and Full-retrain at each testing period on the Yelp data. From the left subfigure (a), we observe that SML outperforms Full-retrain on the two types of new items (OU-NI and NU-NI) by a large margin. From the right subfigure (b), we observe that SML improves over Full-retrain on the type of new user-old item (NU-OI), while achieves a performance comparable with Full-retrain on the type of old user-old item (OU-OI).



(a) Two types of new items



(b) Two types of old items

Figure 6: Recommendation performance of SML and Full-retrain on Yelp grouped by four interaction types: old user-new item (OU-NI), new user-new item (NU-NI), old user-old item (OU-OI), and new user-old item (NU-OI).

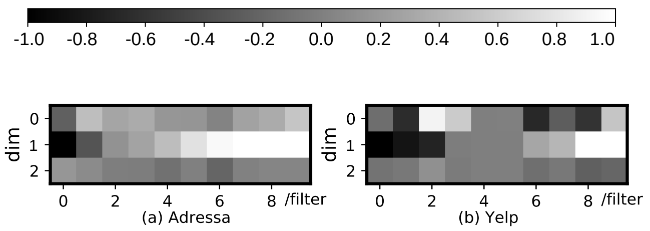


Figure 7: Visualization of the learned filters.

From these results, we draw the conclusion that the improvements of SML over Full-retrain are mainly from the recommendations for new users and new items. This shows the strong ability of SML in quickly adapting to new data that are more reflective of user short-term interests. Moreover, the performance on the interaction type of old user-old item is not degraded, which verifies the effectiveness of SML in capturing long-term interests.

4.5.2 Visualization of CNN Filters. We study the learned CNN filters to disentangle how the transfer fuses W_{t-1} and $\hat{W}_t$. In Figure 7, we visualize the learned filters of the first CNN layer in the item transfer network. Note that similar patterns can be observed in the user transfer network, which are omitted for space. We can see that the filters learned from Yelp and Adressa encode different patterns. For Adressa, the row vector corresponding to $\hat{W}_t$ (*i.e., dim* = 1) has higher values than the other two vectors in general, justifying that the transfer network pays more attention to recent data. For Yelp, we can see two special filters (the 2^{nd} and 8^{th} one), where the values at $dim = 0$ (corresponding to W_{t-1}) and $dim = 1$ have opposite sign. It means that the transfer learns to utilize the difference between $\hat{W}_t$ and W_{t-1}, which is beneficial to capture how user interests evolve.

5 RELATED WORKS

5.1 Recommendation on Sequential Data

The user-item interaction data naturally forms a sequence because each interaction is associated with timestamp information. A large body of work has modeled a sequence of interactions to predict the next interaction, called as sequential [29], next-item/basket [20, 47] or session-based recommendation [18, 46]. An early representative method is Factorized Personalized Markov Chain (FPMC) [34], which models the transition between an interacted item and the previously interacted item with matrix factorization. Later work has extended the first-order modeling [12] to high-order modeling [45]. Recently, many neural network models have been developed, wherein recurrent neural network (RNN) is a natural choice for sequence modeling [18]. The latest work [20] points out a limitation of RNN that it fails to learn the personalized item frequency information in next-basket recommendation. In addition, CNN has also been used for sequential recommendation [38, 46, 47], which is stronger than RNN in modeling the inter-independency between items. Here we consider the sequential nature of interaction data in an orthogonal way — the model needs be periodically retrained to adapt to new data. We propose a general sequential training paradigm, which is technically viable to deploy on the sequential recommendation models.

Another line of work is online or streaming recommendation [3, 16, 37], which aims to refresh recommendations based on real-time user interactions. Several strategies have been proposed, which make different trade-offs between model freshness and accuracy. For example, [5, 16, 25] perform local model updates for each new interaction, which is easy to suffer from forgetting long-term preference when many new interactions are updated. [6, 42] address the issue by sampling a faction of historical interactions and mixing them with new interactions for model updating. The sampler is designed heuristically and needs be adapted manually for different recommendation domains. Our method does not use historical data for model refreshing, achieving good trade-off between long-term and short-term modeling by optimizing for the future performance.

5.2 Meta-Learning

Meta-learning, or learning to learn, aims to quickly and effectively adapt to new tasks by using the prior experience learned from the related tasks [9, 30]. A representative method is Model-Agnostic Meta-Learning (MAML) [9], which represents a general paradigm that uses the testing data of a task to optimize the training process (e.g., initialization and hyper-parameters [10]). The idea of MAML has been taken to solve the cold-start recommendation problem [2, 8, 24, 28, 39]. For example, by treating each user as a task, [2, 24] learn how to generalize well with few iterations. Besides, some works have utilized meta learning to select recommendation algorithms [31] and a recent work [4] proposes λOpt to optimize regularization hyper-parameters based on the validation performance. Inspired by MAML and λOpt, we optimize the model retraining process based on the future validation data, proposing a new training paradigm for sequential user-item interactions.

Beyond recommendation, lifelong learning [23, 27] is weakly relevant to this work, which aims to learn different tasks in a sequence. Several strategies were devised to avoid catastrophically forget old tasks when learning a new task so that all tasks are well served. For instance, [23] remembers old tasks by selectively slowing down learning on the weights important for those tasks. As the key objective of lifelong learning is to serve the current task (on D_t) without suffering performance on previous task, these strategies are not suitable for the sequential training of recommendation model where optimizing the model for better serving the future task (on D_{t+1}) is of importance.

6 CONCLUSION

In this work, we investigated the retraining of recommender models. We formulated the task of recommender retraining, which aims to achieve good generalization on next-period data by modeling the newly collected data. To address the task, we proposed a sequential meta-learning (SML) approach, which consists of 1) an expressive transfer network that converts the previous model to a new model based on the newly collected data, and 2) a sequential training method that effectively utilizes the next-period data to learn the transfer network. We conducted experiments on two datasets of different properties, providing extensive results and analyses on the effectiveness, efficiency, and rationality of SML.

In future, we will extend our generic training paradigm to a wide range of recommender models, such as the recently emerging graph neural networks [14, 44] that are more effective for collaborative filtering, and factorization machines [13, 32] that can incorporate various side information and sequential recommender models [38]. Currently, we do not consider online learning strategy (e.g., bandit methods and local parameter updates) in each serving period, and we plan to take it into account. Lastly, we will develop personalized meta-learning mechanisms that optimize the learning process for different users differently.

REFERENCES

[1] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent cross: Making use of context in recurrent recommender systems. In *WSDM*. 46–54.

[2] Homanga Bharadhwaj. 2019. Meta-Learning for User Cold-Start Recommendation. In *IJCNN*. 1–8.

[3] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A Hasegawa-Johnson, and Thomas S Huang. 2017. Streaming recommender systems. In *WWW*. 525–534.

[4] Yihong Chen, Bei Chen, Xiangnan He, Chen Gao, Yong Li, Jian-Guang Lou, and Yue Wang. 2019. λOpt: Learn to Regularize Recommender Models in Finer Levels. In *SIGKDD*. 978–986.

[5] Robin Devooght, Nicolas Kourtellis, and Amin Mantrach. 2015. Dynamic matrix factorization with priors on unknown values. In *SIGKDD*. 189–198.

[6] Ernesto Diaz-Aviles, Lucas Drumond, Lars Schmidt-Thieme, and Wolfgang Nejdl. 2012. Real-time top-n recommendation in social streams. In *RecSys*. 59–66.

[7] Xiaoyu Du, Xiangnan He, Fajie Yuan, Jinhui Tang, Zhiguang Qin, and Tat-Seng Chua. 2019. Modeling Embedding Dimension Correlations via Convolutional Neural Collaborative Filtering. *TOIS* 37, 4 (2019), 47:1–47:22.

[8] Zhengxiao Du, Xiaowei Wang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Sequential Scenario-Specific Meta Learner for Online Recommendation. In *SIGKDD*. 2895–2904.

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, Vol. 70. 1126–1135.

[10] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. 2018. Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In *ICML*, Vol. 80. 1563–1572.

[11] Jon Atle Gulla, Lemei Zhang, Peng Liu, Özlem Özgöbek, and Xiaomeng Su. 2017. The Adressa dataset for news recommendation. In *WI*. 1042–1048.

[12] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. In *RecSys*. 161–169.

[13] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *SIGIR*. 355–364.

[14] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*.

[15] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*. 173–182.

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

[17] Dan Hendrycks and Kevin Gimpel. 2016. Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. *CoRR* abs/1606.08415 (2016).

[18] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *ICLR*.

[19] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks* 4, 2 (1991), 251–257.

[20] Haoji Hu, Xiangnan He, Jinyang Gao, and Zhi-Li Zhang. 2020. Modeling Personalized Item Frequency Information for Next-basket Recommendation. In *SIGIR*.

[21] Muhammad Abdullah Jamal and Guo-Jun Qi. 2019. Task Agnostic Meta-Learning for Few-Shot Learning. In *CVPR*. 11719–11727.

[22] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[23] James Kirkpatrick, Razvan Pascanu, Neil C Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabskabarwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *PNAS* 114, 13 (2017), 3521–3526.

[24] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. 2019. MeLU: Meta-Learned User Preference Estimator for Cold-Start Recommendation. In *SIGKDD*. 1073–1082.

[25] Wenqiang Lei, Xiangnan He, Yisong Miao, Qingyun Wu, Richang Hong, Min-Yen Kan, and Tat-Seng Chua. 2020. Estimation-Action-Reflection: Towards Deep Interaction Between Conversational and Recommender Systems. In *WSDM*. 304–312.

[26] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *SIGKDD*. 1754–1763.

[27] David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient Episodic Memory for Continual Learning. In *NeurIPS 2017*. 6467–6476.

[28] Feiyang Pan, Shuokai Li, Xiang Ao, Pingzhong Tang, and Qing He. 2019. Warm Up Cold-start Advertisements: Improving CTR Predictions via Learning to Learn ID Embeddings. In *SIGIR*. 695–704.

[29] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-aware recommender systems. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 66:1–66:36.

[30] Sachin Ravi and Hugo Larochelle. 2017. Optimization as a Model for Few-Shot Learning. In *ICLR*.

[31] Yi Ren, Cuirong Chi, and Zhang Jintao. 2019. A Survey of Personalized Recommendation Algorithm Selection Based on Meta-learning. In *CSIA*. 1383–1388.

[32] Steffen Rendle. 2010. Factorization Machines. In *ICDM*. 995–1000.

[33] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*. 452–461.

[34] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *WWW*. 811–820.

[35] Steffen Rendle and Lars Schmidt-Thieme. 2008. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *RecSys*. 251–258.

[36] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. In *NeurIPS*. 2503–2511.

[37] Karthik Subbian, Charu Aggarwal, and Kshiteesh Hegde. 2016. Recommendations for streaming data. In *CIKM*. 2185–2190.

[38] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM*. 565–573.

[39] Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. 2017. A meta-learning perspective on cold-start recommendations for items. In *NeurIPS*. 6904–6914.

[40] Jeffrey S Vitter. 1985. Random sampling with a reservoir. *TOMS* 11, 1, 37–57.

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

[42] Weiqing Wang, Hongzhi Yin, Zi Huang, Qinyong Wang, Xingzhong Du, and Quoc Viet Hung Nguyen. 2018. Streaming ranking based recommender systems. In *SIGIR*. 525–534.

[43] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT: Knowledge Graph Attention Network for Recommendation. In *SIGKDD*. 950–958.

[44] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *SIGIR*. 165–174.

[45] Bin Wu, Xiangnan He, Zhongchuan Sun, Liang Chen, and Yangdong Ye. 2019. ATM: An Attentive Translation Model for Next-Item Recommendation. *IEEE Transactions on Industrial Informatics* 16, 3 (2019), 1448–1459.

[46] Fajie Yuan, Xiangnan He, Haochuan Jiang, Guibing Guo, Jian Xiong, Zhezhao Xu, and Yilin Xiong. 2020. Future Data Helps Training: Modeling Future Contexts for Session-based Recommendation. In *WWW*. 303–313.

[47] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. 2019. A Simple Convolutional Generative Network for Next Item Recommendation. In *WSDM*. 582–590.