Sequential Scenario-Specific Meta Learner for Online Recommendation

Zhengxiao Du[1], Xiaowei Wang[2], Hongxia Yang[2], Jingren Zhou[2], Jie Tang[1]
[1] Department of Computer Science and Technology, Tsinghua University
[2] DAMO Academy, Alibaba Group
duzx16@mails.tsinghua.edu.cn,{daemon.wxw,yang.yhx,jingren.zhou}@alibaba-inc.com,jietang@tsinghua.edu.cn

ABSTRACT

Cold-start problems are long-standing challenges for practical recommendations. Most existing recommendation algorithms rely on extensive observed data and are brittle to recommendation scenarios with few interactions. This paper addresses such problems using *few-shot learning* and *meta learning*. Our approach is based on the insight that having a good generalization from a few examples relies on both a generic model initialization and an effective strategy for adapting this model to newly arising tasks. To accomplish this, we combine the scenario-specific learning with a model-agnostic sequential meta-learning and unify them into an integrated end-to-end framework, namely **S**cenario-specific **S**equential **Meta** learner (or s^2*Meta*). By doing so, our *meta-learner* produces a generic initial model through aggregating contextual information from a variety of prediction tasks while effectively adapting to specific tasks by leveraging learning-to-learn knowledge. Extensive experiments on various real-world datasets demonstrate that our proposed model can achieve significant gains over the state-of-the-arts for cold-start problems in online recommendation. Deployment is at the Guess You Like session, the front page of the Mobile Taobao; and the illustration video can also be watched from the link[1].

CCS CONCEPTS

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

KEYWORDS

Recommender systems, Personalized ranking, Neural network, Meta learning, Few-shot learning

ACM Reference Format:
Zhengxiao Du[1], Xiaowei Wang[2], Hongxia Yang[2], Jingren Zhou[2], Jie Tang[1]
. 2019. Sequential Scenario-Specific Meta Learner for Online Recommendation. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3292500.3330726

[1] https://youtu.be/TNHLZqWnQwc

1 INTRODUCTION

The personalized recommendation is an important method for information retrieval and content discovery in today's information-rich environment. Personalized recommender systems, where the recommendation is generated according to users' past behaviors or profiles, have been proven effective in domains including E-Commerce [29], social networking services [27], video-sharing websites [10], among many others. Traditionally, a personalized recommender system can be seen as a mapping $\mathcal{U} \times \mathcal{I} \to \mathbb{R}$, where $\mathcal{U}$ is the user set and $\mathcal{I}$ is the item set. The mapping result can be a real value for explicit ratings or a binary value for implicit feedback [18, 26, 26, 47]. This setting usually assumes that the behavior pattern of the same user is relatively stationary in different contexts, which is not true in many practical tasks [31, 36]. For example, during the Singles Day Promotion (Double 11) period, the largest online shopping festival in China, consumers sometimes shop impulsively allured by the low discounts. In such scenarios, the contextual information of Double 11 is quite critical. It also has been shown that including contextual information leads to better predictive models and better quality of recommendations [1, 35].

Though context-aware recommender systems have been proven effective [3], they are facing several challenges. Firstly, a large portion of scenarios in a system is actually long-tailed, without enough user feedback. Moreover, the life cycle of a scenario can be quite short. In Taobao, most promotions end within a few days or even a few hours after being launched, without enough time to collect sufficient user feedback for training. Therefore, training scenario-specific recommenders with limited observations are practical requirements. Most existing recommendation algorithms rely on extensive observed data and are brittle to new products and/or consumers [51, 53]. Although the cold-start problem can be tackled by cross-domain recommender systems with domain knowledge being transferred, they still require a large amount of shared samples across domains [21, 32, 43]. Secondly, when training a predictive model on a new scenario, the hyperparameters often have a great influence on the performance and optimal hyperparameters in different scenarios may differ significantly [5, 50]. Finding a right combination of hyperparameters usually requires great human efforts along with sufficient observations.

This paper addresses the problem of cold-start scenario recommendation with the recent progress on *few-shot learning* [44, 52, 55] and *meta-learning* [4, 14, 34, 39]. Our approach is to build a *meta learner* that learns how to instantiate a recommender with good generalization capacity from limited training data. The framework, which generates a scenario-specific recommender by a sequential learning process, is called **S**cenario-specific **S**equential **Meta** learner (or s^2*Meta*). The sequential process, which resembles the traditional machine learning process controlled by human experts, consists of three steps: the meta learner automatically initializes the recommender to be broadly suitable to many scenarios, finetunes its parameters with flexible updating strategy, and stops learning timely to avoid overfitting. The policy of meta learner is learned on a variety of existing scenarios in the system and can guide the quick acquisition of knowledge in new scenarios. By automating the learning process in each scenario, we can also reduce the human efforts needed to train the predictive model on new scenarios that continuously appear in the recommender systems.

Organization. The remainder of this paper is organized as follows: in Section 2, we review related works. Section 3 introduces the problem definition and meta learning settings. Section 4 gives a detailed description of the proposed s^2*Meta* framework. Experimental results on large-scale public datasets and the newly released Taobao theme recommendation dataset are shown in Section 5. At last, Section 6 concludes the paper.

2 RELATED WORK

In this section, we go over the related works on context-aware recommendation, cross-domain recommendation, and meta learning respectively.

2.1 Context-Aware Recommendation

The importance of contextual information has been recognized by researchers and practitioners in many disciplines, including E-Commerce personalization [35], information retrieval [23], data mining [6] and marketing [36], among many others [31, 46]. Relevant contextual information does matter in recommender systems, and it is important to take account of contextual information, such as time, location, or acquaintances' impacts [1, 2, 35]. Compared to traditional recommender systems that make predictions based on the information of users and items, context-aware recommender systems [1, 3] make predictions in the space of $\mathcal{U} \times \mathcal{I} \times C$, where C is the context space. The contextual information can be observable (e.g., time, location, etc), or unobservable (e.g., users' intention). The latter case is related to *Session-based Recommendation* [19] or *Sequence-aware Recommendation* [38]. Even if the contextual information is fully-observable, the weight of different types of information is entirely domain-dependent and quite tricky to be tuned for cold-start scenarios.

2.2 Cross-Domain Recommendation

Traditional recommender systems suggest items belonging to a single domain and this is not perceived as a limitation, but as a focus on a particular market [10, 26, 27]. However, nowadays, users provide feedback for items of different types, express their opinions on different social media and different providers. Providers also wish to cross-sell various categories of products and services, especially to new users. Unlike traditional recommender systems that work on homogeneous users and items, cross-domain recommender systems [8, 13], closely related to Transfer Learning [49], try to combine the information from heterogeneous users and/or items. In [30, 43], they extend the traditional matrix factorization [26] or factorization machines [40] with interacted information from an auxiliary domain to inform recommendations in a target domain. In [32], a multi-layer perceptron is used to capture the nonlinear mapping function across different domains. In [21], they propose a novel neural network CoNet, with architecture designed for knowledge transfer across domains. However, all these methods require a large amount of interacted data of the shared users or items between the source and target domains. In [56], they propose to align the different features in two domains without shared users or items with semantic relatedness. However, they assume that the relatedness of features in the source and target domains can be inferred from domain prior knowledge.

2.3 Meta Learning

Meta learning [7], or *Learning to learn*, which aims to learn a learner that can efficiently complete different learning tasks, has gained great success in few-shot learning settings [14, 39, 44, 52]. The information learned across all tasks guides the quick acquisition of knowledge within each separate learning task. Previous work on meta learning can be divided into two groups. One is the metric method that learns a similarity metric between new instances and instances in the training set. Examples include Siamese Network [25], Matching Network [52], and Prototypical Network [44]. The other is the parameter method that directly predicts or updates the parameters of the classifier according to the training data. Examples include MAML [14], Meta-LSTM [39] and Meta Network [33]. Most methods follow the framework in [52], trying to solve a group of learning tasks of the same pattern, e.g., learning to recognize the objects of different categories with only one example per category given [28]. Previous works on the application of *learning to learn* in recommender systems mainly focus on the algorithm selection [11–13], leaving other parts of the learning process less explored. In [51], they propose to use meta learning to solve the user cold-start problem in the Tweet recommendation. However, they use a neural network to directly output parameters of the recommender, which limits the representation ability of their model.

3 PRELIMINARY

In this section, we formally define the problem of scenario-aware recommendation and introduce the meta learning settings. We summarize the notations in Table 1.

3.1 Problem Formulation

A scenario-based recommender system is represented as $\{\mathcal{U}, \mathcal{I}, C\}$, where $\mathcal{U}$ is the user set, $\mathcal{I}$ is the item set and C is the scenario set. A scenario $c \in C$ is defined as a common context in which users interact with items and $(u, i) \in \boldsymbol{H}_c$ represents that user u interacted with item i in the scenario c. Each scenario c is connected with a recommender function $f_c : \mathcal{U} \times \mathcal{I} \to \mathbb{R}$, where $f_c(u, i)$ is the ranking score of item i for user u in scenario c. For each scenario c, we have access to a training set $D_c^{\text{train}} = \{(u_k, i_k)_{k=1}^{N_c}\}$ where $(u_k, i_k) \in \boldsymbol{H}_c$. Our task is to recommend top-n items for each user u in scenario c and maximize the probability of the user's follow-up behaviors, e.g., clicks or conversions. Notice that, compared to previous works in *cross-domain recommender systems* that require a large amount of training samples on each domain, our work studies the case where the size of D_c^{train} is limited. The latter usually

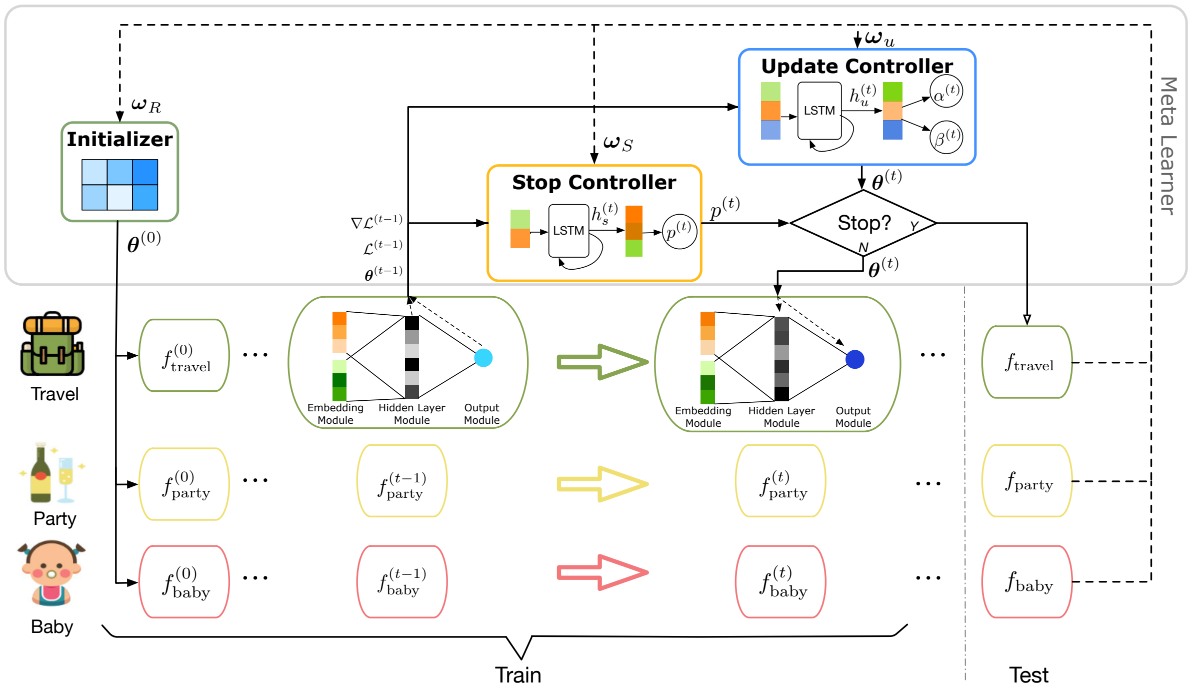

Figure 1: The framework of meta learner and recommender. In each scenario, the recommender is initialized by the initializer and updated by the update controller. After the learning process is stopped by the stop controller, the loss of the final recommender on the test set is computed, and the parameters of meta learner are updated by the meta-gradient.

happens in practice, e.g., new promotion scenarios come out with very sparse related user-item instances.

Table 1: Notations

Notation	Definition or Descriptions
$\mathcal{U}, \mathcal{I}, C$	the user set, item set and scenario set
$m = \lvert\mathcal{U}\rvert, n = \lvert\mathcal{I}\rvert$	numbers of users and items
c	a recommendation scenario
$\boldsymbol{H}_c \subset \mathcal{U} \times \mathcal{I}$	the interaction set of c
T_c	the learning task connected with c
$D_c^{\text{train}}, D_c^{\text{test}}$	the training set and testing set of T_c
$f_c : \mathcal{U} \times \mathcal{I} \to \mathbb{R}$	the recommender function for c
$\boldsymbol{\theta}_c$	parameters of f_c
$\boldsymbol{U}, \boldsymbol{I}$	embedding matrices for users and items
M	meta learner
$\mathbb{T}_{\text{meta-train}}, \mathbb{T}_{\text{meta-test}}$	meta-training set and meta-testing set
$\boldsymbol{\omega} = \{\boldsymbol{\omega}_R, \boldsymbol{\omega}_u, \boldsymbol{\omega}_s\}$	parameters of the meta learner
$\boldsymbol{\omega}_R$	shared initial parameters for f_c
$\boldsymbol{\omega}_u, \boldsymbol{\omega}_s$	parameters of update and stop controllers
$\boldsymbol{\alpha}, \boldsymbol{\beta}$	input gate and forget gate
$p^{(t)}$	stop probability at step t

3.2 Scenario-Specific Meta Learning Settings

The recommendation task in the scenario c can be considered as a learning task, whose training set is D_c^{train}. Our goal is to learn a meta learner M that, given D_c^{train}, predicts parameters of f_c, $\boldsymbol{\theta}_c$.

Similar to the standard few-shot learning settings [39, 52], we assume access to a set of training tasks as the meta-training set, denoted as $\mathbb{T}_{\text{meta-train}}$. Each training task $T_c \in \mathbb{T}_{\text{meta-train}}$ corresponds to a scenario c and has its training/testing pairs: $T_c = \{D_c^{\text{train}}, D_c^{\text{test}}\}$. D_c^{test} is the set of pairwise testing instances: $D_c^{\text{test}} := \{(u, i, i^-)|(u, i) \in \boldsymbol{H}_c \wedge (u, i) \notin D_c^{\text{train}} \wedge (u, i^-) \notin \boldsymbol{H}_c\}$. The meta-training set can be easily built with the previous scenarios in the recommender system, by randomly dividing the user-item interactions in each scenario into the training set and testing set. The parameters $\boldsymbol{\omega}$ of the meta learner is optimized w.r.t. the meta-training objective:

$$\min_{\boldsymbol{\omega}} \mathbb{E}_{T_c}\left[\mathcal{L}_\omega(D_c^{\text{test}}|D_c^{\text{train}})\right], \tag{1}$$

where $\mathcal{L}_\omega(D_c^{\text{test}}|D_c^{\text{train}})$ is the loss on the testing set D_c^{test} given the training set D_c^{train} and meta learner parameters $\boldsymbol{\omega}$. Specifically, we use the hinge loss as the loss function:

$$\mathcal{L}_\omega(D_c^{\text{test}}|D_c^{\text{train}}) = \sum_{(u_k, i_k, i_k^-) \in D_c^{\text{test}}} \frac{\ell(u_k, i_k, i_k^-; \boldsymbol{\theta}_c)}{|D_c^{\text{test}}|}, \tag{2}$$

$$\ell(u, i, i^-; \boldsymbol{\theta}_c) = \max\left(0, \gamma - f(u, i; \boldsymbol{\theta}_c) + f(u, i^-; \boldsymbol{\theta}_c)\right), \tag{3}$$

where the margin γ is set to 1 and $\boldsymbol{\theta}_c = M(D_c^{\text{train}}; \boldsymbol{\omega})$. $\boldsymbol{\theta}_c$ is generated via a sequential process, which we call *scenario-specific learning*. During the scenario-specific learning, the meta learner initializes $\boldsymbol{\theta}_c$ and updates it via gradient descent of flexible steps. After the learning, we will dump the training set and use the recommender f_c for further tasks in the scenario c. During the evaluation, a different set of learning tasks is used, called meta-testing set

$\mathbb{T}_{\text{meta-test}}$, whose scenarios are unseen during meta-training, i.e. $\mathbb{T}_{\text{meta-train}} \cap \mathbb{T}_{\text{meta-test}} = \emptyset$.

4 THE PROPOSED FRAMEWORK

In this section, we detail the modules of the recommender network and meta learner for scenario-specific learning.

4.1 Recommender Network

We apply a feedforward neural network as the recommender f_c, which consists of three modules, to take inputs (u, i) and generate corresponding outputs $f_c(u, i)$.

Embedding Module ($u, i \rightarrow \boldsymbol{e}_u, \boldsymbol{e}_i$): This module consists of two embedding matrices $\boldsymbol{U} \in \mathbb{R}^{m\times d}$ and $\boldsymbol{I} \in \mathbb{R}^{n\times d}$ for users and items respectively. The embeddings can be generated from user/item attributes and/or general interactions without contextual information, with methods in collaborative filtering [26, 41] or network embedding [15, 37]. The user u and item i are first mapped to one-hot vectors $\boldsymbol{x}_u \in \{0,1\}^m$ and $\boldsymbol{x}_i \in \{0,1\}^n$, where only the element corresponding to the user/item id is 1 and all others are 0. The one-hot vectors are then transformed into continuous representations by embedding matrices: $\boldsymbol{e}_u = \boldsymbol{U}\boldsymbol{x}_u$ and $\boldsymbol{e}_i = \boldsymbol{I}\boldsymbol{x}_i$.

Hidden Layer Module ($\boldsymbol{e}_u, \boldsymbol{e}_i \rightarrow \boldsymbol{z}_{ui}$): This module is the central part of the recommender. Similar to deep recommendation models in [9, 10], the user and item embeddings are concatenated as $\boldsymbol{e}_{ui} = [\boldsymbol{e}_u, \boldsymbol{e}_i]$. $\boldsymbol{e}_{ui}$ is then mapped by L hidden layers to a continuous representation of user-item interaction $\boldsymbol{z}_{ui} = \phi_L(\cdots(\phi_1(\boldsymbol{e}_{ui}))\cdots)$. The l-th layer can be represented as:

$$\phi_l(\boldsymbol{z}) = \text{ReLU}(\boldsymbol{W}_l\boldsymbol{z} + \boldsymbol{b}_l), \tag{4}$$

where $\boldsymbol{W}_l$ and $\boldsymbol{b}_l$ are the weight matrix and the bias vector of the l-th layer.

Output Module $\boldsymbol{z}_{ui} \rightarrow f_c(u, i)$: This module computes the recommendation score $f_c(u, i)$ based on the mapped representation of user-item interaction from the last hidden layer. This is achieved by a linear layer as $f_c(u, i) = \boldsymbol{w}^T\boldsymbol{z}_{ui}$, where $\boldsymbol{w}$ is the weight of output layer.

The recommender parameters $\boldsymbol{\theta}_c$, include $\{(\boldsymbol{W}_l, \boldsymbol{b}_l)_{l=1}^L, \boldsymbol{w}\}$, are learned during scenario-specific learning. Note that the embedding matrices $\boldsymbol{U}$ and $\boldsymbol{I}$ are not included, to improve the recommender's generalization ability for unobserved users and items in D_c^{train}. In other words, embedding matrices are shared across different scenarios and kept fixed in scenario-specific learning.

4.2 *s*^{2}*Meta*

In the scenario-specific learning, the recommender parameters $\boldsymbol{\theta}_c$ are learned from the training set D_c^{train}. We summarize the following three challenges in the learning:

- How should the parameters $\boldsymbol{\theta}_c$ be initialized? Randomly initialized parameters can take a long time to converge and often lead to overfitting in the few-shot setting.
- How should the parameters $\boldsymbol{\theta}_c$ be updated w.r.t the loss function? Traditional optimization algorithms rely on carefully tuned hyperparameters to converge to a good solution. Optimal hyperparameters on different scenarios may vary a lot.

Algorithm 1 Training Algorithm of Meta Learner

```
Input: Meta-training set $\mathbb{T}_{\text{meta-train}}$, Loss function $\mathcal{L}$
1:  $\boldsymbol{\omega}_u, \boldsymbol{\omega}_s, \boldsymbol{\omega}_R \leftarrow$ Random Initialization
2:  for $d \leftarrow 1, K$ do        ▷ $K$ is the number of meta-training steps
3:      $D_c^{\text{train}}, D_c^{\text{test}} \leftarrow$ Random scenario from $\mathbb{T}_{\text{meta-train}}$
4:      $\boldsymbol{\theta}_c^{(0)} \leftarrow \boldsymbol{\omega}_R, T \leftarrow 0$
5:      for $t \leftarrow 1, T_{max}$ do
6:          $B^{(t)} \leftarrow$ Random batch from $D_c^{\text{train}}$
7:          $\mathcal{L}^{(t)} \leftarrow \mathcal{L}(B^{(t)}; \boldsymbol{\theta}_c^{(t-1)})$
8:          $p^{(t)} \leftarrow M_s(\mathcal{L}^{(t)}, ||\nabla_{\theta_c^{(t-1)}}\mathcal{L}^{(t)}||_2; \boldsymbol{\omega}_s)$        ▷ Equation (9)
9:          $s^{(t)} \sim$ Bernoulli($p^{(t)}$) ▷ Randomly decide to stop or not
10:         if $s^{(t)} = 1$ then
11:             Break;
12:         end if
13:         $\boldsymbol{\theta}_c^{(t)} \leftarrow$ Update $\boldsymbol{\theta}_c^{(t-1)}$ according to Equations (7) and (8)
14:         $T \leftarrow T + 1$
15:     end for
16:     $\mathcal{L}^{\text{test}} \leftarrow \mathcal{L}(D_c^{\text{test}}; \boldsymbol{\theta}_c^{(T)})$
17:     Update $\boldsymbol{\omega}_u, \boldsymbol{\omega}_R$ using $\nabla_{\omega_u}\mathcal{L}^{\text{test}}$, $\nabla_{\omega_R}\mathcal{L}^{\text{test}}$
18:     $d\boldsymbol{\omega}_s \leftarrow 0$
19:     for $j \leftarrow 1, T$ do
20:         $d\boldsymbol{\omega}_s \leftarrow d\boldsymbol{\omega}_s + (\mathcal{L}^{\text{test}} - \mathcal{L}(D_c^{\text{test}}; \boldsymbol{\theta}_c^{(j)}))\nabla_{\boldsymbol{\omega}_s} \ln(1 - p^{(j)})$
21:     end for
22:     Update $\boldsymbol{\omega}_s$ using $d\boldsymbol{\omega}_s$        ▷ Equation (11)
23: end for
```

- When should the learning process stop? In few-shot setting, learning too much from a small training set can lead to overfitting and hurt the generalization performance.

These challenges are often solved by experts manually in traditional machine learning settings. Instead, we propose *s*^{2}*Meta* which can automatically learn to control the learning process from end to end, including parameter initialization, update strategy and early-stop policy. In the following part, we will introduce how the meta learner controls the three parts of scenario-specific learning and how the meta learner is trained on the meta-training set $\mathbb{T}_{\text{meta-train}}$.

4.2.1 Parameter Initialization. At the beginning of scenario-specific learning, the recommender parameters are initialized as $\boldsymbol{\theta}_c^{(0)}$. Traditionally, the parameters of a neural network are initialized by randomly sampling from a normal distribution or uniform distribution. Given enough training data, the randomly initialized parameters can usually converge to a good local optimum but may take a long time. In the few-shot setting, however, random initialization combined with limited training data can lead to serious overfitting, which hurts the ability of the trained recommender to generalize well. Instead, following [14], we initialize the recommender parameters from the global initial values shared across different scenarios. These initial values are considered as one of meta learner's parameters, denoted as $\boldsymbol{\omega}_R$. Suitable initial parameters may not perform well on a specific scenario, but can adapt quickly to new scenarios given a small amount of training data.

4.2.2 Update Strategy. At each step t, a batch $B^{(t)} = \{u_k, i_k, i_k^-\}_{k=1}^N$ with the fixed size N is sampled from D_c^{train}, where

$(u_k, i_k) \in D_c^{\text{train}}$ and i_k^- is sampled from $\mathcal{I}$ such that $(u_k, i_k^-) \notin D_c^{\text{train}}$. Then the previous parameters $\boldsymbol{\theta}_c^{(t-1)}$ are updated to $\boldsymbol{\theta}_c^{(t)}$ according to $\mathcal{L}^{(t)}$, the loss on $B^{(t)}$:

$$\mathcal{L}^{(t)} = \sum_{(u_k, i_k, i_k^- \in B^{(t)})} \frac{\ell(u_k, i_k, i_k^-; \boldsymbol{\theta}_c^{(t-1)})}{|B^{(t)}|}, \tag{5}$$

where ℓ is the loss function in Equation (3).

The most common method to update parameters of a neural network is stochastic gradient descent (SGD) [42]. In SGD, the parameters $\boldsymbol{\theta}_c$ are updated as:

$$\boldsymbol{\theta}_c^{(t)} = \boldsymbol{\theta}_c^{(t-1)} - \alpha \nabla_{\theta_c^{(t-1)}} \mathcal{L}^{(t)}, \tag{6}$$

where α is the learning rate. There are many variations based on SGD, such as Adam [24] and RMSprop [48], both of which adjust the learning rate dynamically.

Although hand-crafted optimization algorithms have gained success in training deep networks, they rely on a large amount of training data and carefully selected hyperparameters to converge to a good solution. In few-shot learning, the inappropriate learning rate can easily lead to being stuck in a poor local optimum. Moreover, optimal learning rates in different scenarios can differ significantly. Instead, we extend the idea of learning the optimization algorithm in [39] to learn an *update controller* for parameters in $\boldsymbol{\theta}_c$, implementing a more flexible update strategy than hand-crafted algorithms. The update strategy for $\boldsymbol{\theta}_c$ is:

$$\boldsymbol{\theta}_c^{(t)} = \boldsymbol{\beta}^{(t)} \odot \boldsymbol{\theta}_c^{(t-1)} - \boldsymbol{\alpha}^{(t)} \odot \nabla_{\theta_c^{(t-1)}} \mathcal{L}^{(t)}, \tag{7}$$

where $\boldsymbol{\alpha}^{(t)}$ is the *input gate*, similar to the learning rate in SGD; and $\boldsymbol{\beta}^{(t)}$ is the *forget gate*, which can help the meta learner quickly "forget" the previous parameters to leave a poor local optimum.

Since the optimization process is a sequential process, in which the historical information matters, we use LSTM [20] to encode historical information and issue input gates and forget gates:

$$\begin{aligned} \boldsymbol{h}_u^{(t)}, \boldsymbol{c}_u^{(t)} &= \text{LSTM}([\nabla_{\theta_c^{(t-1)}} \mathcal{L}^{(t)}, \mathcal{L}^{(t)}, \boldsymbol{\theta}_c^{(t-1)}], \boldsymbol{h}_u^{(t-1)}, \boldsymbol{c}_u^{(t-1)}), \\ \boldsymbol{\beta}^{(t)} &= \sigma(\boldsymbol{W}_F \boldsymbol{h}_u^{(t)} + \boldsymbol{b}_F), \\ \boldsymbol{\alpha}^{(t)} &= \sigma(\boldsymbol{W}_I \boldsymbol{h}_u^{(t)} + \boldsymbol{b}_I), \end{aligned} \tag{8}$$

where LSTM$(\cdot,\cdot,\cdot)$ represents one-step forward in standard LSTM, $\boldsymbol{h}_u^{(t)}$ and $\boldsymbol{c}_u^{(t)}$ are hidden state and cell state of the update controller at step t, and $\sigma(\cdot)$ is the sigmoid function. The parameters of the updated controller, denoted as $\boldsymbol{\omega}_u$, include $\{\boldsymbol{W}_F, \boldsymbol{b}_F, \boldsymbol{W}_I, \boldsymbol{b}_I\}$ as well as LSTM related parameters. Different parameters in $\boldsymbol{\theta}_c$ correspond to different update controllers. In this way, different learning strategies can be applied.

4.2.3 *Early-Stop Policy.*

In machine learning, overfitting is one of the critical problems that restrict the performance of models. When the model's representation ability exceeds the complexity of the problem, which is often the case for neural networks, the model might fit the sampling variance and random noise in the training data and get poor performance on the testing set. Regularization tricks like L_2 regularizer or dropout[45] are often applied to limit the model's complexity. Another common trick is early-stop: the learning process is stopped when the training loss stops descending or the performance on the validation set begins to drop.

In few-shot setting, as we found, regularizers cannot prevent overfitting to the small training set. Also, as the size of the training set is too small, the validation set divided from the training set cannot provide a precise estimation of generalization ability. To overcome the drawback of hand-crafted stop rules, we propose to learn the stop policy with a neural network, which we call *stop controller* M_s. To balance exploitation and exploration, we apply a stochastic stop policy, in which at step t, the learning process stops with probability $p^{(t)}$, which is predicted by M_s. Similar to the update controller, M_s is an LSTM that can encode historical information:

$$\begin{aligned} \boldsymbol{h}_s^{(t)}, \boldsymbol{c}_s^{(t)} &= \text{LSTM}([\mathcal{L}^{(t)}, ||\nabla_{\theta_c^{(t-1)}} \mathcal{L}^{(t)}||_2], \boldsymbol{h}_s^{(t-1)}, \boldsymbol{c}_s^{(t-1)}), \\ p^{(t)} &= \sigma(\boldsymbol{W}_s \boldsymbol{h}_s^{(t)} + b_s), \end{aligned} \tag{9}$$

where $||\nabla_{\theta_c^{(t-1)}} \mathcal{L}^{(t)}||_2$ is L_2-norm of gradients at step t, and $h_s^{(t)}$ and $c_s^{(t)}$ are the hidden state and the cell state of the stop controller at step t. The parameters of the stop controller, denoted as $\boldsymbol{\omega}_s$, include $\{\boldsymbol{W}_s, \boldsymbol{b}_s\}$ as well as LSTM related parameters.

4.2.4 *Training of Meta Learner.*

The objective of the meta learner is to minimize the expected loss on the testing set after scenario-specific learning on the training set, as is described in Equations (1) to (3). The parameters $\boldsymbol{\omega}$ of the meta learner include shared initial parameters $\boldsymbol{\omega}_R$, parameters of the update controllers $\boldsymbol{\omega}_u$ and parameters of the stop controller $\boldsymbol{\omega}_s$.

The gradients of $\boldsymbol{\omega}_R$ and $\boldsymbol{\omega}_u$ with respect to the meta-testing loss $\mathcal{L}_\omega(D_c^{\text{test}}|D_c^{\text{train}})$, which we call *meta-gradient*, can be computed with back-propagation. However, the meta-gradient may involve higher-order derivatives, which are quite expensive to compute when T is large. Therefore in MAML[14], they only take one-step gradient descent. Instead, our model can benefit from flexible multi-step gradient descent. We ignore higher-order derivatives in the meta-gradient by taking the gradient of the recommender parameters, $\nabla_{\theta_c^{(t-1)}} \mathcal{L}^{(t)}$, as independent of $\boldsymbol{\omega}_R$ and $\boldsymbol{\omega}_u$. With the gradients, we can optimize $\boldsymbol{\omega}_R$ and $\boldsymbol{\omega}_u$ with normal SGD.

Since the relation between $\boldsymbol{\omega}_s$ and $\boldsymbol{\theta}_c$ is discrete and stochastic, it is impossible to take direct gradient descent on $\boldsymbol{\omega}_s$. We use stochastic policy gradient to optimize $\boldsymbol{\omega}_s$. Given one learning trajectory based on stop controller parameters $\boldsymbol{\omega}_s$: $\boldsymbol{\theta}_c^{(0)}, \boldsymbol{\theta}_c^{(1)}, \boldsymbol{\theta}_c^{(2)}, \cdots, \boldsymbol{\theta}_c^{(T)}$, We define the immediate reward at step t as the loss decrease of one-step update: $r^{(t)} = \mathcal{L}(D_c^{\text{test}}; \boldsymbol{\theta}_c^{(t-1)}) - \mathcal{L}(D_c^{\text{test}}; \boldsymbol{\theta}_c^{(t)})$. Then the accumulative reward at t is:

$$Q^{(t)} = \sum_{i=t}^{T} r^{(t)} = \mathcal{L}(D^{\text{test}}; \boldsymbol{\theta}_c^{(t-1)}) - \mathcal{L}(D^{\text{test}}; \boldsymbol{\theta}_c^{(T)}), \tag{10}$$

which is the loss decrease from step t to the end of learning . According to the REINFORCE algorithm[54], we can update $\boldsymbol{\omega}_s$ as:

$$\boldsymbol{\omega}_s \leftarrow \boldsymbol{\omega}_s + \gamma \sum_{t=1}^{T} Q^{(t)} \nabla_{\omega_s} \ln M_s(\mathcal{L}^{(t)}, ||\nabla_{\theta_c^{(t-1)}} \mathcal{L}^{(t)}||_2; \boldsymbol{\omega}_s), \tag{11}$$

where γ is the learning rate for $\boldsymbol{\omega}_s$. The detailed training algorithm for the s^2*Meta* learner is described in Algorithm 1.

5 EXPERIMENT

In this section, we detail the experiments to evaluate the proposed s^2*Meta* in the few-shot scenario-aware recommendation, including details of the datasets, competitors, experimental settings and comparison results. We will also delve deeper to analyze how s^2*Meta* helps the recommender network achieves better results with process sensitivity analyses. Finally, we analyze how the architecture of the recommender influences performance.

5.1 Experimental Settings

5.1.1 Datasets. Current available open datasets for context-aware recommender systems are mostly of small scale with very sparse contextual information [22]. We evaluate our method on two public datasets and one newly released large scenario-based dataset from Taobao[2]. The first dataset is the Amazon Review dataset[17], which contains product reviews and metadata from Amazon. We keep the reviews with ratings no less than three as the positive user-item interactions and take interactions in different categories as different scenarios. Our second dataset is the Movielens-20M [16] dataset, with rating scores from the movie recommendation service Movielens. Following [18], we transform the explicit ratings into implicit data, where all the movies the user has rated are taken as positive items for the user. The tags of movies are used as scenarios. In each dataset, we select scenarios with less than 1,000 but more than 100 items/movies as few-shot tasks with enough interactions for evaluation. Our third dataset is from the click log of Cloud Theme, which is a crutial recommendation procedure in the Taobao app. Different themes correspond to different scenarios of purchase, e.g., *"what to take when traveling" "how to dress up yourself on a party" "things to prepare when a baby is coming"*. In each scenario, a collection of items in related categories is displayed, according to the scenario as well as the user's interests. The dataset includes more than 1.4 million clicks from 355 different scenarios in a 6-days promotion season, with one-month purchase history of users before the promotion started. Table 2 summarizes the statistics of three datasets.

5.1.2 Baselines. We select state-of-the-art baselines in item recommendation in the same domain or cross-domains which can be broadly divided into the following categories:

Heuristic: ItemPop: Items are ranked according to their popularity in specific scenarios, judged by the number of interactions in corresponding training sets. This is a non-personalized baseline in item recommendation[41].

General Domain: NeuMF: The Neural Collaborative Filtering[18] is a state-of-the-art item recommendation method. It combines the traditional MF and MLP in neural networks to predict user-item interactions. This method is not proposed for cross-domain recommendation, and we train a single model on all the scenarios.

[2]Downloadable from https://tianchi.aliyun.com/dataset/dataDetail?dataId=9716.

Shallow Cross-Domain: (1) CDCF: The Cross-Domain Collaborative Filtering[30] is a simple method based on Factorization Machines [40] for cross-domain recommendation. By factorizing the interaction data of shared users in the source domain, it allows extra information to improve recommendation in a target domain; (2) CMF: Collective Matrix Factorization[43] is a matrix factorization method for cross–domain recommendation. It jointly learns low-rank representations for a collection of matrices by sharing common entity factors, which enables the knowledge transfer across domains. We use it as a shallow cross-domain baseline, with users partially overlapping in different domains.

Deep Cross-Domain: (1) EMCDR: The Embedding and Mapping framework for Cross-Domain Recommendation [32] consists of two steps. Step 1 is to learn the user and item embeddings in each domain with latent factor models. Step 2 is to learn the embedding mapping between two domains with a multilayer perceptron from the shared users/items in two domains; (2) CoNet: The Collaborative Cross Networks[21] enables dual knowledge transfer across domains by introducing cross connections from one base network to another and vice versa.

Table 2: Statistics of the Datasets. #Inter. denotes the number of user-item interactions and #Scen. denotes the number of scenarios we use as few-shot tasks.

Dataset	#Users	#Items	#Inter.	#Scen.
Amazon	766,337	492,505	17,523,124	1,289
Movielens	138,493	27,278	20,000,263	306
Taobao	775,603	1,452,525	5,717,835	355

More details in experimental settings can be found in Appendix A.

5.2 Performance Comparison

In this subsection, we report comparison results and summarize insights. Table 3 shows the results on the three datasets concerning top-N recalls [47]. We can see that s^2*Meta* achieves the best results throughout three datasets, compared to both shallow cross-domain models (CMF and CDCF) and deep cross-domain models (EMCDR and CoNet). Involving the extracted scenario information, s^2*Meta* also performs better compared to the general recommendation (NeuMF) and the heuristic method ItemPop.

For the Amazon dataset, s^2*Meta* gives 9.41% improvements on average compared with the best baseline (EMCDR), and achieves 16.47% improvements in terms of Recall@50 compared to the non-scenario-aware NeuMF, which demonstrates the benefits of combining scenario information. Among the cross-domain baselines, neural cross-domain methods are slightly better than the shallow cross-domain methods.

For the Movielens dataset, s^2*Meta* achieves 2.87% improvements on average compared to the best baseline (CoNet). It is a phenomenon common in Amazon and Movielens that domain-specific methods can perform better than the left competitors, showing that the

Table 3: The top-N recall results on test scenarios.

	Amazon			Movielens			Taobao		
Method	Recall@10	Recall@20	Recall@50	Recall@10	Recall@20	Recall@50	Recall@20	Recall@50	Recall@100
NeuMF	24.55	35.65	55.19	31.67	51.30	84.98	25.64	42.31	58.84
ItemPop	26.86	32.65	50.42	39.65	54.32	78.12	18.25	28.57	32.44
CDCF	10.27	16.72	32.79	29.19	41.04	65.75	9.81	20.93	32.14
CMF	27.64	38.31	55.24	29.80	46.91	74.37	9.16	16.47	29.31
EMCDR	31.71	42.14	58.73	43.55	60.89	83.54	20.43	31.52	45.67
CoNet	30.17	41.57	56.06	46.62	63.61	87.06	20.27	31.48	44.53
s^2Meta	**34.39**	**46.53**	**64.28**	**47.79**	**66.02**	**89.07**	**27.11**	**44.10**	**59.98**
Improve	8.35	10.42	9.45	2.51	3.79	2.31	5.73	4.23	1.90

user interests relatively concentrate in a specific scenario. Also, we can find that the deep cross-domain methods can outperform shallow cross-domain methods by improvements over 10%. The performance of NeuMF, which is not designed for cross-domain recommendation, is also superior to CMF and CDCF, showing the power of deep learning in recommendation.

For the Taobao dataset, s^2*Meta* achieves 3.95% performance lifts on average. For this dataset, NeuMF performs best among all the baselines. The reason might be that this dataset is from the click log in the real-world recommendation and clicks often contain more noise than purchase or ratings. The performances of baselines that adapt to the scenario in a naive way might be detrimental by overfitting in the few-shot setting. Above all, s^2*Meta* can still outperform NeuMF with a flexible learning strategy learned by the proposed meta learner.

Note that the relative improvements by taking account of scenario information vary among three datasets. For Amazon, most scenario-specific baselines can perform better than the general domain baseline, while for the Taobao dataset, most scenario-specific baselines cannot outperform NeuMF. It implies that the usefulness of scenario information varies, and s^2*Meta* can dynamically adapt to different datasets' requirements.

5.3 Process Sensitivity Analysis

In this subsection, we analyze how the s^2*Meta* works to control the scenario-specific learning process. Due to the space limit, we only illustrate results from Amazon and performance patterns are the same from the other two datasets.

5.3.1 Impact of Different Parts. First, we analyze the contributions of three parts of meta learner described in Section 4.2. Specifically, we compare the performances of the following variations:

- *Complete:* The complete meta learner that controls parameter initialization, update strategy, and early-stop policy.
- *RandInit:* Parameters of the recommender are randomly initialized. The update strategy and early-stop policy remain unchanged.
- *FixedLr:* Parameters of the recommender are updated by standard SGD, with fixed learning rate 0.01. Parameter initialization and early-stop policy remain unchanged.
- *FixedStep:* The step of scenario-specific learning is fixed at 20, while parameter initialization and update strategy remain consistent with the complete method.

The performance comparison is listed in Table 4. We can see that the complete model can significantly outperform the three weakened variations, indicating that three parts of the meta learner all help to improve the final performance. Among the three variations, the random initialization hurts the performance most. When the parameters are randomly initialized, the recommender has to learn from a random point each time and the learning will take more steps and more easily be stuck in a local optimum. The effect of removing early-stop policy is relatively small. The reason might be that when the learning process is too long, the update controller can still give low input gates to avoid overfitting.

Table 4: Impact of different parts in meta learner on Amazon dataset

Method	Recall@10	Recall@20	Recall@50
RandInit	33.12	45.44	63.35
FixedLr	33.56	45.90	63.86
FixedStep	33.84	46.13	64.02
Complete	**34.39**	**46.53**	**64.28**

5.3.2 Case Study: the learning process learned by the meta learner. To further analyze the learning process the meta learner learned automatically, we select a typical scenario on Amazon and visualize its learning process. Figure 2 shows the training loss, stop probability(predicted by the stop controller), recall on the test set(not visible to the meta learner) and the input and forget gates(predicted by the update controller). From the leftmost figure, we can see that as the training loss ceases to drop, the stop probability rises dramatically, finally leading to the end of the learning process. In fact we can find that the recall on the test set has been stable and further training might lead to overfitting. From the visualization of input gates and forget gates, different update strategies are applied for different layers and different parameters. For different layers, we can find that for the last hidden layer, the input gates remain low and the forget gates remain high, which indicates its parameters remain stable during training. Among different parameters, the

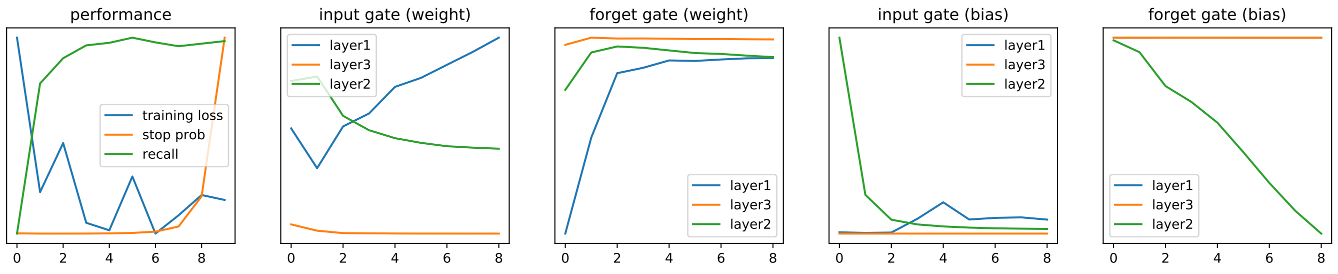

Figure 2: The visualization of the learning process. Left most is the training loss, stop probability and recall on the testing set during the learning process. The other four figures are average input gates and forget gates of weights and biases in the hidden layers.

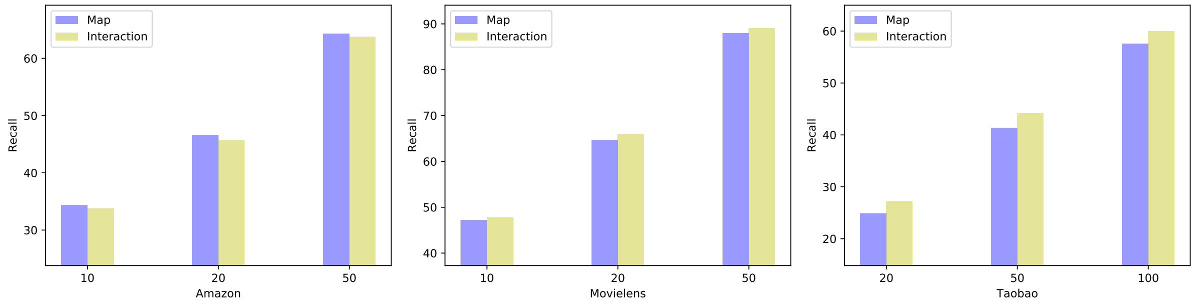

Figure 3: Impact of the recommender architecture on the Amazon(left), Movielens(middle) and Taobao(right) datasets.

weight matrices mainly change at the beginning, and the bias vectors mainly change at the end. This strategy might help to quickly adjust the parameters by updating a part of the parameters at a time. Also, updating the bias vectors can be a minor complement to the updated weight matrices.

5.4 Analysis of Recommender Architecture

In this subsection, we analyze how the architecture of the recommender influences the performance. We compare the following two architectures for the recommender:

- *Mapping Module:* The user and item embeddings are mapped by two multilayer perceptrons respectively, and the final score is computed as the dot product of the mapped user and item embeddings. This is the architecuture of EMCDR.
- *Interaction Module:* User and item embeddings are concatenated and a multilayer perceptron computes the final score from the concatenated vector. This is the architecture used by NeuMF and CoNet.

We compare the performance of two architectures with the same meta learner. The results on three datasets are shown in Figure 3. In general, we can see that in general the interaction module can perform better than the mapping module, because the interaction module can represent the complicated interactions between users and items, while the mapping module only maps the user and item embeddings independently. On Taobao, the relative improvement of interaction modules can reach 6.71%. However, on Amazon, the mapping module slightly outperforms the interaction module. This implies that the optimal recommender architecture may differ among datasets. It's also noted that in practice the mapping module can be more efficient because given the recommender, the mapped embeddings can be pre-computed and the matching can be optimized with advanced data structures.

6 CONCLUSION

In this work, we explored few-shot learning for recommendation in the scenario-specific setting. We proposed a novel sequential scenario-specific framework for recommender systems using meta learning, to solve the cold-start problem in some recommendation scenarios. s^2*Meta* can automatically control the learning process to converge to a good solution and avoid overfitting, which has been the critical issue of few-shot learning. Experiments on real-world datasets demonstrate the effectiveness of the proposed method, by comparing with shallow/deep, general/scenario-specific baselines. In the future, we will automate the architecture design of recommenders. In our experiments, we found that the performance of the same architecture might differ in different datasets. By learning to choose the optimal recommender architecture, the performance of s^2*Meta* can be further improved.

ACKNOWLEDGMENTS

Jie Tang and Hongxia Yang are the corresponding authors of this paper. The work is supported by the NSFC for Distinguished Young Scholar (61825602), Tsinghua University Initiative Scientific Research Program, and a research fund supported by Alibaba.

REFERENCES

[1] Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen, and Alexander Tuzhilin. 2005. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.* 23, 1 (2005), 103–145.

[2] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Trans. Knowl. Data Eng.* 17, 6 (2005), 734–749.

[3] Gediminas Adomavicius and Alexander Tuzhilin. 2015. Context-Aware Recommender Systems. In *Recommender Systems Handbook*. 191–226.

[4] Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *NIPS*. 3981–3989.

[5] Parivash Ashrafi, Yi Sun, Neil Davey, Rod Adams, Marc B. Brown, Maria Prapopoulou, and Gary P. Moss. 2015. The importance of hyperparameters selection within small datasets. In *IJCNN*. 1–8.

[6] Michael J. A. Berry and Gordon Linoff. 1997. *Data mining techniques - for marketing, sales, and customer support.* Wiley.

[7] Pavel Brazdil, Christophe Giraud-Carrier, Carlos Soares, and Ricardo Vilalta. 2008. *Metalearning: Applications to Data Mining* (1 ed.). Springer Publishing Company, Incorporated.

[8] Iván Cantador, Ignacio Fernández-Tobías, Shlomo Berkovsky, and Paolo Cremonesi. 2015. Cross-Domain Recommender Systems. In *Recommender Systems Handbook*. 919–959.

[9] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *RecSys Workshop*. 7–10.

[10] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *RecSys*. 191–198.

[11] Tiago Cunha, Carlos Soares, and Andr C.P.L.F. de Carvalho. 2018. Metalearning and Recommender Systems. *Inf. Sci.* 423, C (Jan. 2018), 128–144.

[12] Tiago Cunha, Carlos Soares, and André C. P. L. F. de Carvalho. 2016. Selecting Collaborative Filtering Algorithms Using Metalearning. In *ECML/PKDD*. 393–409.

[13] Michael D. Ekstrand and John Riedl. 2012. When recommenders fail: predicting recommender failure for algorithm selection and combination. In *RecSys*. 233–236.

[14] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML*. 1126–1135.

[15] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1025–1035.

[16] F. Maxwell Harper and Joseph A. Konstan. 2016. The MovieLens Datasets: History and Context. *TiiS* 5, 4 (2016), 19:1–19:19.

[17] Ruining He and Julian McAuley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *WWW*. 507–517.

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

[19] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. In *CIKM*. 843–852.

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

[21] Guangneng Hu, Yu Zhang, and Qiang Yang. 2018. CoNet: Collaborative Cross Networks for Cross-Domain Recommendation. In *CIKM*. 667–676.

[22] Sergio Ilarri, Raquel Trillo Lado, and Ramón Hermoso. 2018. Datasets for Context-Aware Recommender Systems: Current Context and Possible Directions. In *ICDE Workshops*. 25–28.

[23] Peter Ingwersen and Kalervo Järvelin. 2005. Information Retrieval in Context: IRiX. *SIGIR Forum* 39, 2 (Dec. 2005), 31–39.

[24] Diederick P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

[25] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, Vol. 2.

[26] Yehuda Koren, Robert M. Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer* 42, 8 (2009), 30–37.

[27] Su Mon Kywe, Ee-Peng Lim, and Feida Zhu. 2012. A Survey of Recommender Systems in Twitter. In *SocInfo*. 420–433.

[28] Fei-Fei Li, Robert Fergus, and Pietro Perona. 2006. One-Shot Learning of Object Categories. *IEEE Trans. Pattern Anal. Mach. Intell.* 28, 4 (2006), 594–611.

[29] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* 7, 1 (2003), 76–80.

[30] Babak Loni, Yue Shi, Martha Larson, and Alan Hanjalic. 2014. Cross-Domain Collaborative Filtering with Factorization Machines. In *ECIR*. 656–661.

[31] Denis A. Lussier and Richard W. Olshavsky. 1979. Task Complexity and Contingent Processing in Brand Choice. *Journal of Consumer Research* 6, 2 (1979), 154–165.

[32] Tong Man, Huawei Shen, Xiaolong Jin, and Xueqi Cheng. 2017. Cross-Domain Recommendation: An Embedding and Mapping Approach. In *IJCAI*. 2464–2470.

[33] Tsendsuren Munkhdalai and Hong Yu. 2017. Meta Networks. In *ICML*. 2554–2563.

[34] Alex Nichol, Joshua Achiam, and John Schulman. 2018. On First-Order Meta-Learning Algorithms. *CoRR* abs/1803.02999 (2018).

[35] Cosimo Palmisano, Alexander Tuzhilin, and Michele Gorgoglione. 2008. Using Context to Improve Predictive Modeling of Customers in Personalization Applications. *IEEE Trans. Knowl. Data Eng.* 20, 11 (2008), 1535–1549.

[36] John W. Payne, James R. Bettman, Eloise Coupey, and Eric J. Johnson. 1992. A constructive process view of decision making: Multiple strategies in judgment and choice. *Acta Psychologica* 80, 1 (1992), 107 – 141.

[37] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *SIGKDD*. 701–710.

[38] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-aware Recommender Systems. In *UMAP*. 373–374.

[39] Sachin Ravi and Hugo Larochelle. 2017. Optimization as a model for few-shot learning. In *ICLR*, Vol. 2. 6.

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

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

[42] Herbert Robbins and Sutton Monro. 1951. A Stochastic Approximation Method. *Ann. Math. Statist.* 22, 3 (09 1951), 400–407.

[43] Ajit Paul Singh and Geoffrey J. Gordon. 2008. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*. 650–658.

[44] Jake Snell, Kevin Swersky, and Richard S. Zemel. 2017. Prototypical Networks for Few-shot Learning. In *NIPS*. 4080–4090.

[45] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *JMLR* 15, 1 (Jan. 2014), 1929–1958.

[46] Kostas Stefanidis, Evaggelia Pitoura, and Panos Vassiliadis. 2007. A context-aware preference database system. *Int. J. Pervasive Computing and Communications* 3, 4 (2007), 439–460.

[47] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *WSDM*. 565–573.

[48] Tijmen Tieleman and Geoffrey Hinton. 2012. *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.* Technical Report.

[49] Lisa Torrey and Jude Shavlik. 2009. Transfer Learning.

[50] Jan N. van Rijn and Frank Hutter. 2018. Hyperparameter Importance Across Datasets. In *SIGKDD*. 2367–2376.

[51] Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. 2017. A Meta-Learning Perspective on Cold-Start Recommendations for Items. In *NIPS*. 6907–6917.

[52] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. 2016. Matching Networks for One Shot Learning. In *NIPS*. 3630–3638.

[53] Maksims Volkovs, Guang Wei Yu, and Tomi Poutanen. 2017. DropoutNet: Addressing Cold Start in Recommender Systems. In *NIPS*. 4964–4973.

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

[55] Wenhan Xiong, Mo Yu, Shiyu Chang, Xiaoxiao Guo, and William Yang Wang. 2018. One-Shot Relational Learning for Knowledge Graphs. In *EMNLP*. 1980–1990.

[56] Deqing Yang, Jingrui He, Huazheng Qin, Yanghua Xiao, and Wei Wang. 2015. A Graph-based Recommendation across Heterogeneous Domains. In *CIKM*. 463–472.

A APPENDIX

In this section, we first give the deployment description and implementation notes of our proposed models. The implementation notes and parameter configurations of compared methods are then given. Finally, we introduce the experiment settings in three datasets.

A.1 Deployment

Deployment is at the Guess You Like session, the front page of the Mobile Taobao; and the illustration video can also be watched from the link[3].

A.2 Implementation Notes

The architecture of the recommender network is designed as three hidden layers and one output layer. The embedding size is set to 128 on Amazon and Alibaba Theme and 64 on Movielens. The number of hidden units in each layer is half of that in the last layer. The hidden size of the update LSTM is set to 16. The hidden size of the stop LSTM is set to 18. For the input of the update controllers and stop controller, we use the preprocessing trick in [4]:

$$x \longrightarrow \begin{cases} (\frac{\log(|x|)}{p}, \operatorname{sgn}(x)) & \text{if } |x| \geq e^{-p} \\ (-1, e^{p}x) & \text{otherwise} \end{cases} \tag{12}$$

where $p = 10$ in our experiment. s^2*Meta* is trained by standard SGD with learning rate 0.0001 and weight decay 1e-5, implemented with PyTorch[4] 1.0 in Python 3.6 and runs on a single Linux server with 8 NVIDIA GeForce GTX 1080.

A.3 Compared Methods

A.3.1 Code. The code of NeuMF is provided by the author[5]. We simply adapt the input and evaluation modules to our experimental settings. For CMF, we adpot the implementation of LIBMF[6]. For CDCF, we dapot the official libFM implementation[7]. The codes of EMCDR and CoNet are not published and we implement them with PyTorch.

A.3.2 Parameter Configuration. We do grid search over the hyperparameters of CDCF, CMF, EMCDR, and CoNet, with the meta-training set. The search space for CDCF and CMF is learning rate $(0.1, 0.03, 0.001)$, learning step $(10, 100, 200)$ and factor dimension $(8, 16, 32)$. The search space for EMCDR is learning rate $(0.1, 0.01, 0.001, 0.0001)$ and learning step $(10, 100, 1000)$. The search space for CoNet is learning rate $(0.01, 0.001)$, learning step $(10, 50, 100, 500)$ and sparse ratio λ $(0.1, 0.01, 0.001)$. For NeuMF, we use the pre-training suggested by the author, with embedding size of MF model set to 8 and MLP layers set to $[64, 32, 16, 8]$.

A.4 Experiment Setting

As we mention in Section 5.1.1, it is challenging to find large-scale public datasets to evaluate context-aware recommender systems. Therefore we evaluate our method on two public datasets in the cross-domain setting. We also use one scenario-based dataset in Taobao.

On the Amazon Review[8] dataset, we take user purchases in different leaf categories as different scenarios. To maximize the difference between the source domain and the target domain and fully evaluate the model's ability to learn user behaviors in new scenarios, we select the leaf categories in different first-order categories as the source domain, meta-training set and meta-testing set. The specific split of first-order categories is displayed in Table 5.

On the Movielens-20M[9] dataset, we take the movie tags in the Tag Genome included in the dataset as scenarios. The tag genome is a data structure that contains tag relevance scores for movies that are computed based on user-contributed content. Tags with relevance scores higher than 0.8 to a movie are considered as tags of the movie. We use the movies without tags as the source domain, and randomly divide the tags into the meta-training set and meta-testing set.

On both datasets, we select the scenarios with 100-1000 items/movies as the few-shot tasks with enough interactions for evaluation. Unlike traditional cross-domain settings in which the interactions on different domains are simply concatenated, we are more interested in the cold-start scenarios, on which most users have no previous interactions. Therefore on each scenario, 64 interactions of shared users are used as D_c^{train} and all others are used for evaluation. We use all the interactions on the source domain to generate the user embeddings and the interactions of the users only appearing on the scenario to generate the item embeddings, with Matrix Factorization[26]. For cross-domain baselines, each scenario is considered as a target domain. The training data is all the interactions on the source domain, the few-shot interactions of shared users D_c^{train}, and all the interactions of the users only appearing on the target domain. Therefore, all the comparative methods use the same training data and evaluation set, and the fairness of comparison is guaranteed.

On the Taobao Themes[10] dataset, we take the different themes as different scenarios and all the clicks are considered as positive user-item interactions. We use the one-month purchase history of users before the click log as the source domain and randomly divide the scenarios into the meta-training set and meta-testing set. As this dataset is quite sparse, we use the general user and item embeddings generated from attributes and interactions in Taobao by GraphSage[15]. This embeddings are also used by NeuMF, EMCDR and CoNet.

Table 5: Dataset split on Amazon

Domain	First-order Category
Source Domain	Books, Electronics, Movies and TV Clothing, Shoes and Jewelry, Home and Kitchen Sports and Outdoors, Health and Personal Care Toys and Games, Apps for Android Grocery and Gourmet Food
Meta-training	Beauty, CDs and Vinyl, Kindle Store Tools and Home Improvement, Pet Supplies
Meta-testing	Digital Music, Musical Instruments Video Games, Cell Phones and Accessories Baby, Automotive, Office Products Amazon Instant Video, Patio, Lawn and Garden

[3] https://youtu.be/TNHLZqWnQwc
[4] https://pytorch.org/
[5] http://github.com/hexiangnan/neural_collaborative_filtering
[6] https://www.csie.ntu.edu.tw/~cjlin/libmf/
[7] http://www.libfm.org
[8] http://jmcauley.ucsd.edu/data/amazon
[9] https://grouplens.org/datasets/movielens/
[10] https://tianchi.aliyun.com/dataset/dataDetail?dataId=9716.