Meta-Learning for Online Update of Recommender Systems

Minseok Kim[1], Hwanjun Song[2], Yooju Shin[1], Dongmin Park[1], Kijung Shin[1], Jae-Gil Lee[1]*

[1]KAIST, [2]NAVER AI Lab,
{minseokkim, yooju.shin, dongminpark, kijungs, jaegil}@kaist.ac.kr,
hwanjun.song@navercorp.com

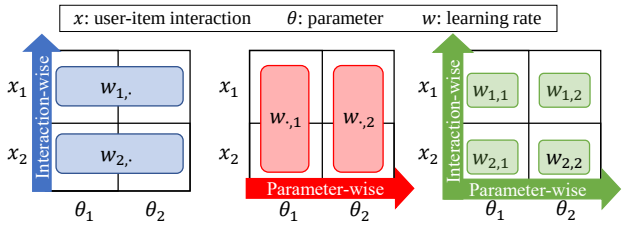
Abstract

Online recommender systems should be always aligned with users' current interest to accurately suggest items that each user would like. Since user interest usually *evolves* over time, the update strategy should be *flexible* to quickly catch users' current interest from continuously generated new user-item interactions. Existing update strategies focus either on the importance of each user-item interaction or the learning rate for each recommender parameter, but such *one*-directional flexibility is insufficient to adapt to varying relationships between interactions and parameters. In this paper, we propose MeLON, a meta-learning based novel online recommender update strategy that supports *two*-directional flexibility. It is featured with an *adaptive* learning rate for each *parameter-interaction pair* for inducing a recommender to quickly learn users' up-to-date interest. The procedure of MeLON is optimized following a meta-learning approach: it learns how a recommender learns to generate the optimal learning rates for future updates. Specifically, MeLON first enriches the meaning of each interaction based on previous interactions and identifies the role of each parameter for the interaction; and then combines these two pieces of information to generate an adaptive learning rate. Theoretical analysis and extensive evaluation on three real-world online recommender datasets validate the effectiveness of MeLON.

Introduction

The widespread of mobile devices enables a large number of users to connect to a variety of online services, such as video streaming (Davidson et al. 2010), shopping (Linden, Smith, and York 2003), and news (Gulla et al. 2017), where each user seeks only a few items out of a myriad of items in services. To keep users involved, online services struggle to meet each user's needs *accurately* by deploying personalized recommender systems (Koren 2008), which suggest the items potentially interesting to him/her. In an online setting where a user's *current* interest changes constantly, the online recommender should catch up each user's up-to-date interest to prevent its service from being stale (He et al. 2016). To this end, recommender models are updated continuously in response to new user-item interactions.

In modern online recommender systems, *fine-tuning* has been widely employed to update models since it is infeasible to re-train the models from scratch whenever new user-item interactions come in. Specifically, pre-trained models (i.e., snapshots trained on past user-item interactions) are fine-tuned based *only* on new user-item interactions. Fine-tuning not only requires less computational cost but also has sufficient capability to reflect up-to-date information (Zhang et al. 2020). However, because *few-shot* incoming user-item interactions are very sparse in the user-item domain (Finn et al. 2019), the standard fine-tuning scheme would not suit online recommender systems to *quickly* adapt to up-to-date user interest. Therefore, the key challenge is to overcome this data sparsity for fine-tuning.



(a) Importance Reweighting. (b) Meta-Optimization. (c) Ours: MeLON.

Figure 1: Flexibility comparison of adjusting a learning rate w to update a parameter θ given a user-item interaction x. While (a) importance reweighting and (b) meta-optimization support only one of the two learning perspectives, (c) MeLON supports both of them.

To cope with this challenge, previous researches have been actively studied in two directions.

- **Importance reweighting** adjusts the importance of *each* new user-item interaction (He et al. 2016; Shu et al. 2019). These methods receive the *loss* of a new user-item interaction from a recommender as a supervisory signal and then determine how much the recommender should be fine-tuned by each user-item interaction.
- **Meta-optimization** controls how much *each* recommender parameter should be fine-tuned from new user-item interactions (Zhang et al. 2020). These methods determine a parameter-wise optimization strategy such as a learning rate, given the loss of the new user-item interactions and each parameter's previous optimization history.

These two orthogonal approaches focus on different aspects on learning: importance reweighting focuses on the impact of each *user-item interaction*, as shown in Figure 1(a), while meta-optimization focuses on that of each *pa-*

*Jae-Gil Lee is the corresponding author.

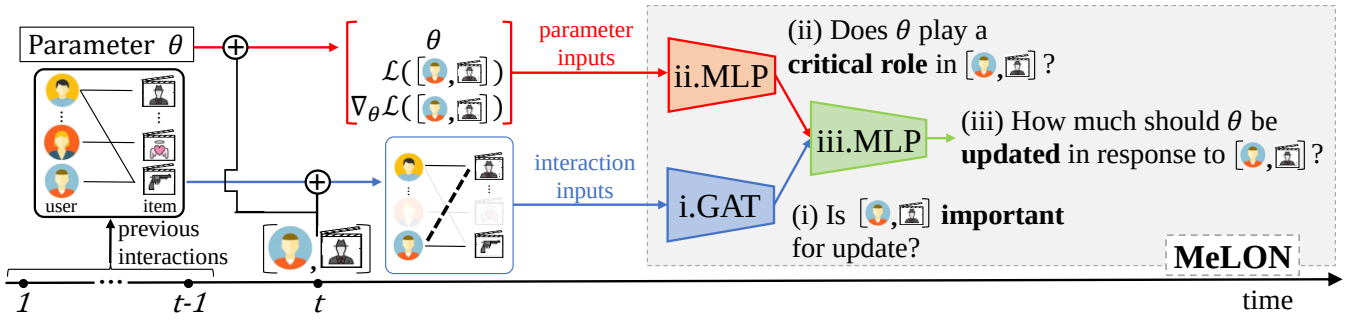


Figure 2: Illustration of MeLON's procedure. MeLON (i) represents the importance of a given user-item interaction for the current update based on previous interactions, (ii) identifies the role of each parameter for the interaction, and (iii) adapts a learning rate specific to each parameter-interaction pair.

rameter, as shown in Figure 1(b). That is, both approaches support the *one*-directional flexibility in the either data or parameter perspective. If we regard fine-tuning at each time period as a distinct task (Zhang et al. 2020), the role of each parameter in a recommender varies for different user-item interactions, because it is well known that an explanatory factor (i.e., parameter) in a neural network has different relevance toward different tasks (Bengio, Courville, and Vincent 2013). Thus, we contend that the flexibility in *both* data and parameter perspectives should be achieved. However, the two existing approaches lack this level of flexibility, possibly leading to sub-optimal recommendation quality.

In this paper, we propose, **MeLON** (Meta-Learning for ONline recommender update), a novel online recommender update strategy that supports the flexibility in *both* data and parameter perspectives. It learns to *adaptively* adjust the learning rate of each parameter for each new user-item interaction, as illustrated in Figure 1(c). Because the optimality of a learning rate depends on how much informative on both perspectives and how to exploit them, we derive three research objectives: (i) how to describe the importance of the task with a new user-item interaction, (ii) how to identify the role of each parameter for the task, then (iii) how to determine the optimal learning rate for each pair of interactions and parameters based on their mutual relevance.

Corresponding to the three research questions, MeLON goes through the following three steps, as shown in Figure 2. First, because exploiting the connections from the new user-item interaction is very helpful to mitigate the *data sparsity* issue, MeLON employs a graph attention network (GAT) (Veličković et al. 2018) to represent the importance of each new user-item interaction along with previous user-item interactions. Then, an explicit neural mapper dynamically captures the role of a parameter, assessing its contribution to the new user-item interaction by the loss and gradient. Last, the two representations—for an interaction and a parameter—are jointly considered to generate the optimal learning rate specific to the interaction-parameter pair. MeLON repeats the three steps for every online update, following the *learning-to-learn* philosophy of meta-learning. That is, the meta-model MeLON learns to provide the learning rates such that the recommender model updated using those learning rates quickly grasps what users want now and succeeds recommendations for them in the future.

The effectiveness of MeLON is extensively evaluated on two famous recommender algorithms using three real-world online service datasets in a comparison with six update strategies. In short, the results show that MeLON successfully improves the recommendation accuracy by up to 29.9% in term of HR@5. Such capability of MeLON is empowered by two-directional flexibility under learning-to-learn strategy, which is further supported by the theoretical analysis and ablation study.

Preliminary and Related Work

Online recommenders build a pre-trained model using previous user-item interactions, and the pre-trained model is continuously updated in response to incoming user-item interactions. A deep neural network (DNN) is widely used for a recommender, and it is updated for each *mini-batch* (Ruder 2016) to quickly adapt to users' up-to-date interest. Let $x = (t, u, i)$ denote a user-item interaction between a user u and an item i at time t. Suppose that a mini-batch $\mathcal{B}_t$ consists of n new user-item interactions at time t. Then, the recommender at time t, parameterized by $\Theta_t = \{\theta_{t,1}, \dots, \theta_{t,M}\}$ where M is the total number of parameters, is updated by

$$\begin{aligned}\Theta_{t+1} &= \Theta_t - \eta \nabla_{\Theta_t} \sum_{x \in \mathcal{B}_t} \frac{1}{n} \mathcal{L}_{\Theta_t}(x) \\ &= \Theta_t - \nabla_{\Theta_t} \mathcal{L}_{\Theta_t}(\mathcal{B}_t)^\top \boldsymbol{W}.\end{aligned} \tag{1}$$

Here, η is a learning rate, and $\mathcal{L}_{\Theta_t}(\mathcal{B}_t) \in \mathbb{R}^n$ denotes the loss of new user-item interactions in the mini-batch $\mathcal{B}_t$ by the recommender model Θ_t under any objective function $\mathcal{L}$ such as mean squared error (MSE) or Bayesian personalized ranking (BPR) (Rendle et al. 2009). The *learning rate matrix* $\boldsymbol{W} \in \mathbb{R}^{n \times M}$ is used to represent the learning rate for a parameter θ_m in response to each user-item interaction x, where all the learning rates (i.e., all elements of $\boldsymbol{W}$) are typically set equally to $w = \frac{\eta}{n}$. Then, the overall performance is derived by evaluating each recommender snapshot for a given mini-batch at each time step,

$$\min_{\{\Theta_t\}_{t=1}^T} \sum_{t=1}^{T} \sum_{x \in \mathcal{B}_t} \mathcal{L}_{\Theta_t}(x) = \min_{\{\Theta_t\}_{t=1}^T} \sum_{t=1}^{T} \mathcal{L}_{\Theta_t}(\mathcal{B}_t). \tag{2}$$

The two directions—importance reweighting and meta-optimization—for online recommender updates are characterized by the construction of the learning rate matrix $\boldsymbol{W}$.

Importance Reweighting

Instead of assigning the equal importance $1/n$ to each user-item interaction as in Eq. (1), *importance reweighting* (He et al. 2016; Shu et al. 2019) assigns a different importance determined by a reweighting function $\phi^I(\cdot)$,

$$\begin{aligned}\Theta_{t+1} &= \Theta_t - \eta \nabla_{\Theta_t} \sum_{x \in \mathcal{B}_t} \mathcal{L}_{\Theta_t}(x) \cdot \underbrace{\phi^I\big(\mathcal{L}_{\Theta_t}(x)\big)}_{\text{interaction-wise}} \\ &= \Theta_t - \nabla_{\Theta_t} \mathcal{L}_{\Theta_t}(\mathcal{B}_t)^\top \boldsymbol{W}^I,\end{aligned} \tag{3}$$

where $\phi^I(\cdot)$ receives the loss of each user-item interaction as its input. That is, $\boldsymbol{W}^I \in \mathbb{R}^{n \times M}$ is constructed such that each row has the same value returned by $\phi^I(\cdot)$. The representative methods differ in the detail of $\phi^I(\cdot)$, as follows:

- eALS (He et al. 2016) applies a heuristic rule that assigns a weight for each new user-item interaction. Typically, a high weight is set to learn the current user interest.
- MWNet (Shu et al. 2019) maintains an external meta-model that adaptively assesses the importance of a given user-item interaction for model update that lets the updated model minimize the loss on meta-data (e.g., next recommendation in online update).

However, this scheme does not support the varying role of a parameter for different tasks.

Meta-Optimization

On the other hand, *meta-optimization* (Ravi and Larochelle 2017; Li et al. 2017; Du et al. 2019; Zhang et al. 2020) aims at adjusting the learning rate of each recommender parameter $\theta_{t,m}$ via a learning rate function $\phi^P(\cdot)$,

$$\begin{aligned}\Theta_{t+1} &= \Theta_t - \underbrace{\phi^P\Big(\mathcal{L}_{\Theta_t}(\mathcal{B}_t), \Theta_t\Big)}_{\text{parameter-wise}} \cdot \nabla_{\Theta_t} \sum_{x \in \mathcal{B}_t} \frac{1}{n} \mathcal{L}_{\Theta_t}(x) \\ &= \Theta_t - \nabla_{\Theta_t} \mathcal{L}_{\Theta_t}(\mathcal{B}_t)^\top \boldsymbol{W}^P,\end{aligned} \tag{4}$$

where the function $\phi^P(\cdot)$ receives the training loss of a mini-batch and the recommender parameters Θ_t as its input. That is, $\boldsymbol{W}^P \in \mathbb{R}^{n \times M}$ is constructed such that each column has the same value returned by $\phi^P(\cdot)$. Again, the representative algorithms differ in the detail of $\phi^P(\cdot)$, as follows:

- S^{2}Meta (Du et al. 2019) exploits MetaLSTM (Ravi and Larochelle 2017) to decide how much to forget a parameter's previous knowledge and to learn new user-item interactions via the gating mechanism of LSTM (Hochreiter and Schmidhuber 1997).
- MetaSGD (Li et al. 2017) maintains one learnable parameter for each model parameter to adjust its learning rate based on the loss.
- SML (Zhang et al. 2020) maintains a convolutional neural network (CNN)-based meta-model with pretrained and fine-tuned parameters. It decides how much to combine the knowledge for previous interactions and that for new user-item interactions for each parameter.

Contrary to importance reweighting, this scheme does not support the varying importance of user-item interactions.

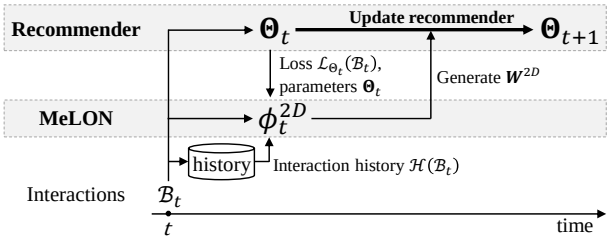


Figure 3: Online update procedure with the meta-model MeLON involved.

Difference from Previous Work

While previous update strategies achieve only *one*-directional flexibility, i.e., $\phi^{1D} \in \{\phi^I, \phi^P\}$, we aim at developing an online update strategy ϕ^{2D} that provides *two*-directional flexibility for the learning rates to be adaptive in *both* data and parameter perspectives,

$$\begin{aligned}\Theta_{t+1} &= \Theta_t - \nabla_{\Theta_t} \sum_{x \in \mathcal{B}_t} \mathcal{L}_{\Theta_t}(x) \cdot \underbrace{\phi^{2D}\big(x, \mathcal{L}_{\Theta_t}(x), \Theta_t\big)}_{\text{interaction-/parameter-wise}} \\ &= \Theta_t - \nabla_{\Theta_t} \mathcal{L}_{\Theta_t}(\mathcal{B}_t)^\top \boldsymbol{W}^{2D},\end{aligned} \tag{5}$$

where the function receives an individual user-item interaction, the training loss of the interaction, and the recommender parameters Θ_t as its input, which are essential ingredients to be adaptive to both user-item interactions and parameters. That is, $\boldsymbol{W}^{2D} \in \mathbb{R}^{n \times M}$ is constructed such that each entry can be filled with a different value returned by $\phi^{2D}(\cdot)$ even when either a user-item interaction or parameter is identical to other entries as in Figure 1(c),

$$\begin{aligned}&(x = x') \wedge (\theta_m \neq \theta_{m'}) \\ &\nRightarrow \phi^{2D}\big(x, \mathcal{L}_{\Theta_t}(x), \theta_m\big) = \phi^{2D}\big(x', \mathcal{L}_{\Theta_t}(x'), \theta_{m'}\big),\end{aligned} \tag{6}$$

$$\begin{aligned}&(x \neq x') \wedge (\theta_m = \theta_{m'}) \\ &\nRightarrow \phi^{2D}\big(x, \mathcal{L}_{\Theta_t}(x), \theta_m\big) = \phi^{2D}\big(x', \mathcal{L}_{\Theta_t}(x'), \theta_{m'}\big).\end{aligned} \tag{7}$$

Methodology: MeLON

MeLON is a *meta-model* that determines the optimal learning rate for each recommender parameter regarding a user-item interaction. Figure 3 shows the collaboration between a recommender model and the meta-model MeLON. For each iteration of online update, a recommender model provides its parameters Θ_t and the loss $\mathcal{L}_{\Theta_t}(\mathcal{B}_t)$ of the current batch $\mathcal{B}_t$ to MeLON; then, additionally using the previous interaction history, MeLON provides the learning rate matrix $\boldsymbol{W}^{2D}$, which is learned to reduce $\mathcal{L}_{\Theta_t}(\mathcal{B}_t)$ as much as possible, to the recommender model; finally, the recommender model is updated using $\boldsymbol{W}^{2D}$ for $\mathcal{B}_t$. Please refer to the Section A of the supplementary material for more details. The internal procedure of MeLON is described according to the three research questions: (i) representing the relevance between a user and an item for each user-item interaction, (ii) representing the role of each parameter for a user-item interaction, and (iii) determining the learning rate for each pair of user-item interactions and parameters.

Step I: Representing User-Item Interaction

Because a single user-item interaction may not contain sufficient information, we utilize the information from the previous interaction history by adding the users and items connected to the user-item interaction. More specifically, the latent representation of the user-item interaction is derived using a graph attention network (GAT) on the bipartite graph that represents the interactions between users and items received until the current time. The bipartite graph for a user-item interaction x is constructed from the users and items in the *interaction history* in Definition 1.

Definition 1. (INTERACTION HISTORY) *Given a user-item interaction $x = (t, u, i)$, the* user interaction history of u *is the set of items interacted with u before t, $\mathcal{H}_{user}(x) = \{i' \mid \exists(t', u, i') \in \mathcal{X} \text{ s.t. } t' < t\}$, where $\mathcal{X}$ is the entire set of user-item interactions; similarly, the* item interaction history of i *is the set of users interacted with i before t, $\mathcal{H}_{item}(x) = \{u' \mid \exists(t', u', i) \in \mathcal{X} \text{ s.t. } t' < t\}$.* □

For the bipartite graph, the users in $\mathcal{H}_{user}(x)$ constitute the user side, and the items in $\mathcal{H}_{item}(x)$ constitute the item side. Here, each user (or item) node is represented by the user (or item) embedding used in the recommender model. An edge is created for each of the previous user-item interactions, and its weight is determined by the attention score between them. Then, a user (or item) embedding is *extended* using the connections to the other side on the bipartite graph, as specified in Definition 2.

Definition 2. (EXTENDED EMBEDDING) *Given a user-item interaction $x = (t, u, i)$, let $\mathbf{e}_u$ and $\mathbf{e}_{i'}$ be the embeddings of u and $i' \in \mathcal{H}_{user}(x)$. Then, the* extended embedding *of u, $\tilde{\mathbf{e}}_u$, is defined as*

$$\tilde{\mathbf{e}}_u = \text{ReLU}(W_{user} \cdot [\mathbf{e}_u, \sum_{i' \in \mathcal{H}_{user}(x)} \alpha_{ui'} \mathbf{e}_{i'}] + \mathbf{b}_{user}), \quad (8)$$

where W_{user} and $\mathbf{b}_{user}$ are a learnable weight matrix and a bias vector. Here, $\alpha_{ui'}$ indicates the attention score for i' and is derived by the GAT, as follows:

$$\alpha_{ui'} = \text{softmax}\big(\text{LeakyReLU}\big([\mathbf{e}_u, \mathbf{e}_{i'}]^\top \mathbf{a}_U\big)\big), \quad (9)$$

where $\mathbf{a}_U$ is a learnable attention vector. In addition, the extended embedding *of an item i, $\tilde{\mathbf{e}}_i$, is defined in the same way to the opposite direction.* □

Last, the two extended embeddings, $\tilde{\mathbf{e}}_u$ and $\tilde{\mathbf{e}}_i$, are concatenated and gone through a linear mapping to learn the relevance between the user and the item, as specified in Definition 3. As a result, the *interaction representation* contains not only the rich information about a user and an item but also the relevance between them.

Definition 3. (INTERACTION REPRESENTATION) *Given a user-item interaction $x = (t, u, i)$, let $\tilde{\mathbf{e}}_u$ and $\tilde{\mathbf{e}}_i$ be the extended embeddings of u and i, respectively. The* interaction representation *of x, $\mathbf{h}_x$, is defined by*

$$\mathbf{h}_x = \text{ReLU}\big(W_x \cdot [\tilde{\mathbf{e}}_u, \tilde{\mathbf{e}}_i] + \mathbf{b}_x\big), \quad (10)$$

where W_x and $\mathbf{b}_x$ are a learnable weight matrix and a bias vector. □

Step II: Representing Parameter Role

Because it is well known that a parameter in a neural network has different relevance toward different tasks (user-item interactions in our study) (Bengio, Courville, and Vincent 2013), we contend that a parameter has a different role for each user-item interaction. The *role* of a parameter can be roughly defined as its degree of impact on users (or items) of common characteristics. For example, a specific parameter may have a high impact on action films, while another parameter may have a high impact on romance films.

To help find a parameter role, the latent representation of a parameter is derived using three types of information: the current value of a parameter $\theta_{t,m}$, the loss $\mathcal{L}_{\Theta_t}(x)$ of a recommender model for a given user-item interaction x, and the gradient $\nabla_{\theta_{t,m}} \mathcal{L}_{\Theta_t}(x)$ of the loss with respect to the parameter. The loss represents how much the recommender model parameterized by Θ_t has not learned that user-item interaction. Each gradient represents how much the corresponding parameter needs to react to that loss; we expect that relevant parameters usually have higher gradients than irrelevant ones. Thus, putting them together, they can serve as useful information for determining a parameter role.

Symmetric to the interaction representation in Definition 3, the *role representation* is obtained through a multi-layer perceptron (MLP), as specified in Definition 4. Because the magnitude of the loss and gradient varies across the pair of interactions and parameters, we apply a preprocessing technique (Ravi and Larochelle 2017; Andrychowicz et al. 2016) to adjust the scale of the loss and gradient as well as to separate their magnitude and sign. As a result, the output of the MLP, $\mathbf{h}_{\theta_{t,m}}$, is regarded to represent the role of $\theta_{t,m}$ with respect to the given user-item interaction x.

Definition 4. (ROLE REPRESENTATION) *Given a parameter $\theta_{t,m}$ and a user-item interaction x, the* role representation *of $\theta_{t,m}$, $\mathbf{h}_{\theta_{t,m}}$ is defined by*

$$\mathbf{h}_{\theta_{t,m}} = \text{MLP}(\big[\theta_{t,m},\ \mathcal{L}_{\Theta_t}(x),\ \nabla_{\theta_{t,m}} \mathcal{L}_{\Theta_t}(x)\big]), \quad (11)$$

where the MLP consists of L linear mapping layers each followed by the ReLU activation function. □

Step III: Adapting Learning Rate

The resulting two representations, $\mathbf{h}_x$ and $\mathbf{h}_{\theta_{t,m}}$, respectively, contain rich information about the importance of the user-item interaction x and the role of the parameter $\theta_{t,m}$. Hence, we employ a linear mapping layer which fuses the two representations and adapts the learning rate $w^{lr}_{x,\theta_{t,m}}$ to the given interaction-parameter pair, as follows:

$$w^{lr}_{x,\theta_{t,m}} = \sigma(W_{lr} \cdot [\mathbf{h}_x, \mathbf{h}_{\theta_{t,m}}] + \mathbf{b}_{lr}), \quad (12)$$

where σ is a sigmoid function. The learning rate is likely to be high if the user-item interaction is important while the parameter plays a key role to the interaction, so that the parameter is quickly adapted to the interaction. Then, the learning rate is used to update the current parameter $\theta_{t,m}$ for the user-item interaction x, as follows:

$$\theta_{t+1,m} = \theta_{t,m} - w^{lr}_{x,\theta_{t,m}} \cdot \nabla_{\theta_{t,m}} \mathcal{L}_{\Theta_t}(x). \quad (13)$$

Theoretical Analysis on Update Flexibility

Our suggested online update strategy MeLON, ϕ^{2D}, leaves a question of how much benefit it can bring compared with the previous two strategies ϕ^I and ϕ^P. As an effort to resolve it, we present a theoretical analysis of the advantage of flexible update in terms of rank, where the rank of a learning rate matrix $rank(\boldsymbol{W})$ demonstrates how flexible an update can be via $\boldsymbol{W}$. That is, when its rank can be higher, $\boldsymbol{W}$ can support more flexible updates of parameters in response to new interactions. The previous strategies limit the rank to 1 since, as discussed above, they provide an identical learning rate either to every interaction or to every parameter. In MeLON, the rank can be higher since the learning rates are adapted to each interaction-parameter pair. In this regard, we show that previous strategies may suffer from large optimality gap with an optimal learning rate matrix $\boldsymbol{W}^*$, while the gap can be reduced by MeLON.

We denote the recommender parameters updated by $\boldsymbol{W}$ as $\hat{\Theta}$ and the optimal parameters as Θ^*. Then, the optimality gap between the two sets of parameters $\|\Theta^* - \hat{\Theta}\|_2$ is dependent on the gap between the learning rate matrices $\|\boldsymbol{W}^* - \boldsymbol{W}\|_2$ in terms of spectral norm as follows:

$$\begin{aligned}\|\Theta^* - \hat{\Theta}\|_2 &= \|(\Theta - \nabla_\Theta \mathcal{L} \cdot \boldsymbol{W^*}) - (\Theta - \nabla_\Theta \mathcal{L} \cdot \boldsymbol{W})\|_2 \\ &= \| - (\nabla_\Theta \mathcal{L} \cdot \boldsymbol{W^*}) - (-\nabla_\Theta \mathcal{L} \cdot \boldsymbol{W})\|_2 \\ &= \|\nabla_\Theta \mathcal{L} \cdot (\boldsymbol{W^*} - \boldsymbol{W})\|_2 \\ &\leq \|\nabla_\Theta \mathcal{L}\|_2 \cdot \|(\boldsymbol{W^*} - \boldsymbol{W})\|_2.\end{aligned}$$

Then, a lower bound of $\|\boldsymbol{W}^* - \boldsymbol{W}\|_2$ is obtained from the singular values σ of $\boldsymbol{W}^*$, as formalized in Lemma 1.

Lemma 1. (Eckart and Young 1936) *Given $\boldsymbol{W}^*$ with its singular value decomposition $\boldsymbol{U\Sigma V}$ and $k \in \{1, \cdots, rank(\boldsymbol{W}^*) - 1\}$, let $\boldsymbol{W}_k^* = \sum_{r=1}^{k} \sigma_r \boldsymbol{U}_r \boldsymbol{V}_r$, where σ_r is the r-th largest singular value. Then, $\boldsymbol{W}_k^*$ is the best rank-k approximation of $\boldsymbol{W}^*$ in terms of spectral norm,*

$$\min_{\boldsymbol{W}: rank(\boldsymbol{W})=k} \|\boldsymbol{W}^* - \boldsymbol{W}\|_2 = \|\boldsymbol{W}^* - \boldsymbol{W}_k^*\|_2 = \sigma_{k+1}.$$

Proof. See (Eckart and Young 1936). □

Based on Lemma 1, we show that a flexible update strategy ϕ^{2D} can enjoy smaller optimality gap than the previous strategies, which are one-directionally flexible.

Lemma 2. *For $\boldsymbol{W}^I$ and $\boldsymbol{W}^P$ (see Eq. (3) and Eq. (4)), $rank(\boldsymbol{W}^I) = rank(\boldsymbol{W}^P) = 1$ holds.*

Proof. Every column of $\boldsymbol{W}^I$ equals to $\phi^I\big(\mathcal{L}_{\Theta_t}(x)\big) \in \mathcal{R}^n$, and every row of $\boldsymbol{W}^P$ equals to $\phi^P\big(\mathcal{L}_{\Theta_t}(\mathcal{B}_t), \Theta_t\big) \in \mathcal{R}^M$. Hence, $rank(\boldsymbol{W}^I) = rank(\boldsymbol{W}^P) = 1$ holds. □

Theorem 3. *For $\boldsymbol{W}^{1D} \in \{\boldsymbol{W}^I, \boldsymbol{W}^P\}$ (see Eq. (3) and Eq. (4)) and $\boldsymbol{W}^{2D}$ (see Eq. (5)), the following inequality holds:*

$$\min_{\boldsymbol{W}^{1D}} \|\boldsymbol{W}^* - \boldsymbol{W}^{1D}\|_2 \geq \min_{\boldsymbol{W}^{2D}} \|\boldsymbol{W}^* - \boldsymbol{W}^{2D}\|_2.$$

Proof. Lemma 1, Lemma 3, and $\boldsymbol{W}^{1D} \in \{\boldsymbol{W}^I, \boldsymbol{W}^P\}$ imply $\min_{\boldsymbol{W}^{1D}} \|\boldsymbol{W}^* - \boldsymbol{W}^{1D}\| = \sigma_2$. On the other hand, $rank(\boldsymbol{W}^{2D}) \geq 1$,[1] Thus, by Lemma 1, $\min_{\boldsymbol{W}^{2D}} \|\boldsymbol{W}^* - \boldsymbol{W}^{2D}\| \leq \sigma_2$, which concludes the proof, holds. □

[1]Specifically, by Eq. (6) and Eq. (7), $rank(\boldsymbol{W}^{2D})$ is not necessarily one and can be greater than one.

Dataset	Users	Items	Interactions
Adressa	29,589	1,457	1,191,114
Amazon	91,013	118,031	3,625,349
Yelp	60,543	74,249	2,880,520

Table 1: Summary of the three real-world datasets.

In the experiments, we empirically validate the advantage of the *two*-directional flexibility of $\boldsymbol{W}^{2D}$.

Evaluation

Our evaluation was conducted to support the following:

- The performance improvement by MeLON is consistent for various datasets and recommenders.
- MeLON helps recommenders quickly adapt to users' up-to-date interest over time.
- The two-directional flexibility in MeLON is very effective for recommendation.
- The training overhead of MeLON is affordable.

Experiment Settings

Datasets. We used three real-world online recommendation benchmark datasets: Adressa (Gulla et al. 2017), Amazon (Ni, Li, and McAuley 2019), and Yelp[2], as summarized in Table 1. The duration that a user's interest persists varies across datasets; relatively short duration for news in Adressa, typically longer duration for locations in Yelp, and in-between them for products in Amazon.

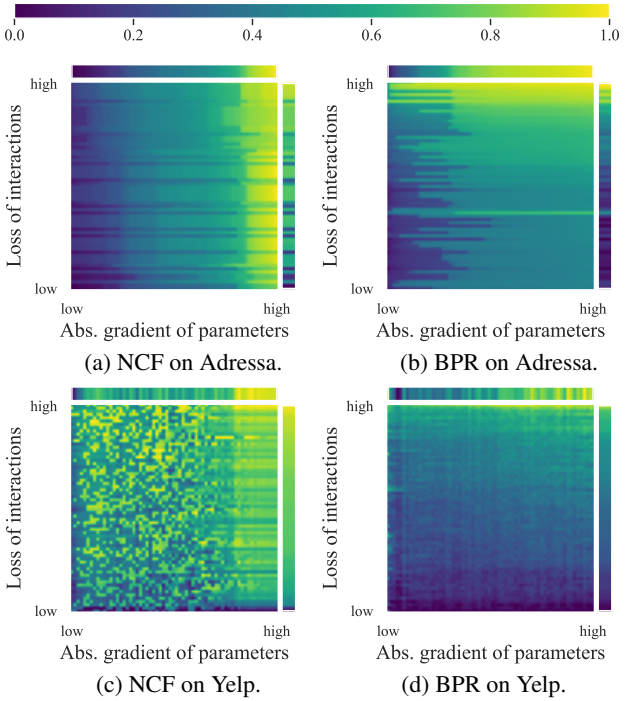
Algorithms and Implementation Details. For the base recommender, we used two popular personalized recommender algorithms: BPR (Koren, Bell, and Volinsky 2009; Rendle et al. 2009) and NCF (He et al. 2017). For the online training strategy, we compared MeLON with six update methods, namely Default, eALS (He et al. 2016), MWNet (Shu et al. 2019), MetaSGD (Li et al. 2017), S^{2}Meta (Du et al. 2019), and SML (Zhang et al. 2020). "Default" is the standard fine-tuning strategy, and the remaining methods are based on either importance reweighting or meta-optimization. Hence, 14 combinations of two recommenders and seven update strategies were considered for evaluation. The experiment setting was exactly the same for all the combinations.

Evaluation Metrics. We used two widely-used evaluation metrics, hit rate (HR) and normalized discounted cumulative gain (NDCG). Given a recommendation list, HR measures the rate of true user-interacted items in the list, while NDCG additionally considers the ranking of the user-interacted items. The two metrics were calculated for top@5, top@10, and top@20 items, respectively. For each mini-batch on prequential evaluation, a recommender estimates the rank of each user's 1 interacted item and randomly-sampled 99 non-interacted items, and this technique is widely used in the literature (He et al. 2016; Du et al. 2019) because it is time-consuming to rank all non-interacted items.

[2]https://www.kaggle.com/yelp-dataset/yelp-dataset

Dataset	Method	NCF						BPR					
		HR@5	HR@10	HR@20	NDCG@5	NDCG@10	NDCG@20	HR@5	HR@10	HR@20	NDCG@5	NDCG@10	NDCG@20
Adressa	Default	0.334±0.007	0.407±0.009	0.502±0.017	0.283±0.005	0.306±0.006	0.330±0.007	0.292±0.002	0.359±0.002	0.422±0.001	0.250±0.001	0.272±0.001	0.288±0.001
	eALS	0.664±0.006	0.750±0.006	0.826±0.004	0.542±0.004	0.570±0.004	0.589±0.003	0.443±0.008	0.520±0.011	0.613±0.016	0.371±0.006	0.396±0.007	0.419±0.008
	MWNet	0.325±0.009	0.392±0.010	0.480±0.014	0.276±0.007	0.297±0.007	0.319±0.008	0.289±0.001	0.356±0.001	0.421±0.008	0.248±0.001	0.269±0.001	0.285±0.000
	MetaSGD	0.275±0.000	0.406±0.002	0.686±0.002	0.229±0.000	0.270±0.000	0.340±0.000	0.276±0.002	0.405±0.002	0.686±0.002	0.230±0.001	0.271±0.001	0.340±0.001
	S^2Meta	0.276±0.016	0.396±0.039	0.548±0.062	0.221±0.008	0.260±0.015	0.298±0.021	0.278±0.015	0.401±0.037	0.559±0.060	0.223±0.007	0.262±0.014	0.302±0.020
	SML	N/A	N/A	N/A	N/A	N/A	N/A	0.270±0.001	0.330±0.002	0.399±0.001	0.235±0.001	0.255±0.001	0.272±0.001
	MeLON	**0.863±0.004**	**0.954±0.000**	**0.982±0.000**	**0.626±0.011**	**0.656±0.010**	**0.664±0.010**	**0.877±0.004**	**0.958±0.001**	**0.983±0.000**	**0.671±0.007**	**0.698±0.006**	**0.705±0.005**
Amazon	Default	0.168±0.001	0.244±0.002	0.359±0.006	0.115±0.001	0.140±0.001	0.168±0.001	0.246±0.002	0.339±0.003	0.457±0.003	0.172±0.002	0.202±0.002	0.231±0.001
	eALS	0.219±0.009	0.323±0.014	0.462±0.019	0.148±0.006	0.182±0.007	0.216±0.008	0.327±0.002	0.425±0.002	0.542±0.001	0.238±0.002	0.270±0.002	0.299±0.001
	MWNet	0.169±0.002	0.247±0.005	0.366±0.010	0.116±0.001	0.142±0.002	0.171±0.003	0.244±0.001	0.339±0.002	0.456±0.003	0.171±0.001	0.201±0.001	0.231±0.001
	MetaSGD	0.151±0.003	0.214±0.005	0.317±0.006	0.104±0.002	0.125±0.003	0.150±0.002	0.148±0.002	0.214±0.005	0.314±0.005	0.103±0.001	0.123±0.002	0.148±0.001
	S^2Meta	0.292±0.005	0.390±0.007	0.497±0.010	0.192±0.003	0.224±0.004	0.250±0.004	0.270±0.029	0.367±0.037	0.474±0.040	0.178±0.001	0.209±0.021	0.237±0.216
	SML	N/A	N/A	N/A	N/A	N/A	N/A	0.220±0.001	0.307±0.001	0.409±0.001	0.153±0.001	0.181±0.001	0.207±0.000
	MeLON	**0.324±0.040**	**0.519±0.034**	**0.807±0.014**	**0.225±0.031**	**0.287±0.028**	**0.360±0.020**	**0.363±0.016**	**0.506±0.013**	**0.650±0.007**	**0.248±0.012**	**0.294±0.011**	**0.330±0.009**
Yelp	Default	0.659±0.001	0.816±0.002	0.923±0.002	0.477±0.001	0.528±0.001	0.555±0.002	0.600±0.003	0.766±0.004	0.883±0.005	0.426±0.002	0.480±0.002	0.509±0.002
	eALS	0.618±0.009	0.781±0.014	0.901±0.001	0.438±0.006	0.491±0.007	0.521±0.001	**0.677±0.003**	**0.831±0.003**	**0.922±0.002**	**0.488±0.002**	**0.538±0.002**	**0.562±0.002**
	MWNet	0.658±0.002	0.818±0.005	0.926±0.003	0.475±0.001	0.527±0.002	0.555±0.002	0.603±0.001	0.771±0.001	0.890±0.001	0.428±0.000	0.483±0.000	0.513±0.000
	MetaSGD	0.207±0.003	0.309±0.005	0.433±0.018	0.136±0.002	0.169±0.003	0.200±0.005	0.209±0.000	0.321±0.003	0.451±0.001	0.137±0.000	0.174±0.001	0.206±0.002
	S^2Meta	0.393±0.005	0.525±0.007	0.654±0.113	0.277±0.003	0.320±0.004	0.351±0.086	0.208±0.001	0.323±0.001	0.471±0.002	0.137±0.000	0.174±0.001	0.211±0.001
	SML	N/A	N/A	N/A	N/A	N/A	N/A	0.478±0.000	0.614±0.000	0.720±0.000	0.338±0.000	0.382±0.000	0.409±0.000
	MeLON	**0.779±0.010**	**0.923±0.003**	**0.980±0.006**	**0.563±0.027**	**0.610±0.024**	**0.624±0.020**	0.619±0.017	0.779±0.016	0.886±0.011	0.439±0.014	0.491±0.014	0.519±0.012

Table 2: Overall online recommendation performance.[3] The average of five executions with the standard error are reported. The best results are marked in bold, and the second best results are underlined.



(a) NCF on Adressa. (b) BPR on Adressa.

(c) NCF on Yelp. (d) BPR on Yelp.

Figure 4: Learning rates obtained by MeLON for interaction-parameter pairs. MeLON tends to provide higher learning rates for the pairs with large losses and gradients.

Please see Section B of the supplementary material for more details of the experiment settings.

Overall Performance Comparison

Table 2 shows the top-k recommendation performance with varying update strategies for the three datasets. Overall, MeLON greatly boosts the recommendation performance compared with the other update strategies in general. It outperforms the other update strategies with NCF in terms of HR@5 by up to 29.9%, 10.9%, and 18.2% in Adressa, Amazon, and Yelp, respectively. This benefit is attributed to the two-directional flexibility of MeLON, which successfully adapts learning rates on each interaction-parameter pair. That is, MeLON considers the importance of the user-item interaction as well as the role of the parameter, while the compared strategies consider only either of them.

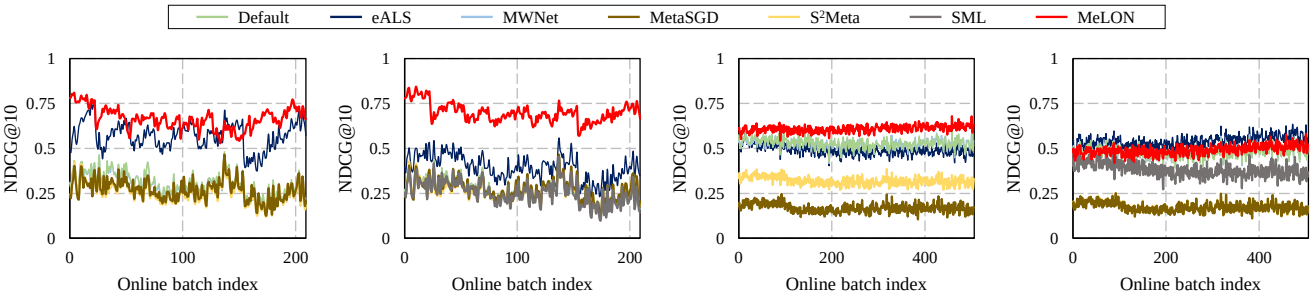
Flexibility Gap. Figure 4 displays the learning rate matrix of all interaction-parameter pairs in MeLON when trained on Adressa and Yelp. Here, each square matrix displays the two-directional learning rates ($\boldsymbol{W}^{2D}$) for all interaction-parameter pairs. On the other hand, the top and right bars are the averages along one axis, which can be considered as the one-directional learning rates ($\boldsymbol{W}^{1D}$). The learning rates in the square matrix are flexibly determined for each interaction-parameter pair, and high learning rates are assigned when the loss or gradient is high. A row or column does *not* stick to the same learning rate, and this visualization clearly demonstrates the necessity of the two-directional flexibility. The gap between the learning rate in $\boldsymbol{W}^{2D}$ and that in $\boldsymbol{W}^{1D}$ in Figure 4 is directly related to how quickly a recommender adapts to up-to-date user interests, which in turn leads to the performance difference between MeLON and the previous update strategies.

In-Depth Comparison. We provide interesting observations for the online update strategies:

- Importance reweighting works well for datasets where a user's interest changes slowly (e.g., Yelp);
- Meta-optimization works well for datasets where a user's interest changes quickly (e.g., Adressa);
- MeLON works well for both types of datasets.

Specifically, in terms of HR@20, an importance reweighting strategy, MWNet, enhances the recommendation performance in Yelp, but shows worse performance in Adressa than Default. In contrast, an opposite trend is observed for the two meta-optimization strategies, MetaSGD and S^2Meta. Thus, we conjecture that, for time-sensitive user interest, such as news in Adressa, it is more important to focus on the parameter roles, which could be associated

[3] SML cannot be implemented on NCF because the transfer network of SML is intended to work on embedding parameters which do not exist in NCF.



(a) NCF on Adressa. (b) BPR on Adressa. (c) NCF on Yelp. (d) BPR on Yelp.

Figure 5: Recommendation performance over each online batch in the Adressa and Yelp datasets.

Dataset	MeLON$_I$	MeLON$_O$	**MeLON**
Adressa	0.293 ± 0.003	0.278 ± 0.015	**0.877 ± 0.004**
Amazon	0.248 ± 0.002	0.269 ± 0.029	**0.363 ± 0.016**
Yelp	0.605 ± 0.001	0.208 ± 0.000	**0.619 ± 0.017**

Table 3: Ablation on MeLON components. HR@5 and its standard error for BPR are reported. MeLON$_I$ and MeLON$_P$ are the variants with only importance reweighting and meta-optimization, respectively.

with the topics in this dataset. On the other hand, for time-insensitive user interest, such as places in Yelp, it would be better to focus on the interaction itself. This claim can be further supported by the different trends in Figure 4, where horizontal (i.e., parameter-wise) lines are more visible in the Adressa dataset, but vertical (i.e., interaction-wise) lines are more visible in the Yelp dataset.

Performance Trend over Time

Figure 5 shows the NDCG@10 performance trend of seven update strategies over each online test batch of the Adressa and Yelp datasets. Overall, only MeLON consistently adheres to the highest performance (close to the highest in Figure 5(d)) compared with other update strategies during the entire test period. The performance gap between MeLON and the others widens especially in Addressa because its news data becomes quickly outdated and needs more aggressive adaptation for better recommendation. In this regard, eALS also shows better performance than others since it always assigns high weights for new user-item interactions. On the other hand, in the Yelp dataset where the user's interest may not change quickly, all the update strategies show small performance fluctuations.

Ablation Study on Two-Directional Flexibility

We conduct an ablation study to examine the two-directional flexibility of MeLON by using its two variants with *partial* flexibility: (i) MeLON$_I$, an importance reweighting variant without parameter-wise inputs and (ii) MeLON$_P$, a meta-optimization variant without interaction-wise inputs. Table 3 shows the performance of the two variants along with the original MeLON on the three datasets. Of course, MeLON is far better than the variants. In addition, the performance of MeLON$_I$ is similar to those of the importance reweighting strategies (e.g., MWNet) in Table 2, while MeLON$_P$ shows

Dataset	Model	Default	eALS	MWNet	MetaSGD	SML	S^{2}Meta	MeLON
Adressa	NCF	0.011	0.011	0.023	0.022	N/A	0.317	0.257
	BPR	0.011	0.011	0.012	0.014	0.163	0.112	0.076
Amazon	NCF	0.010	0.010	0.026	0.022	N/A	2.922	0.263
	BPR	0.009	0.009	0.014	0.012	0.130	1.630	0.072
Yelp	NCF	0.008	0.008	0.016	0.018	N/A	3.345	0.258
	BPR	0.007	0.007	0.008	0.011	0.051	0.431	0.041

Table 4: Elapsed time (sec) of seven update strategies per online batch.

the results similar to the meta-optimization strategies (e.g., S^{2}Meta). Therefore, the power of MeLON is attained when the two-directional flexibility is accompanied.

Elapsed Time for Online Update

Table 4 shows the average elapsed time of the seven update strategies per online batch update. Overall, all the update strategies including MeLON show affordable update time except S^{2}Meta which consumes even seconds in Yelp and Amazon. That is, MeLON is still capable of handling multiple recommender updates within a second, which is fast enough for online recommender training. The speed of MeLON is improved by its *selective* parameter update; given a user-item interaction, MeLON updates only the parameters involved with the recommender's computation for the interaction. This technique helps MeLON maintain its competitive update speed, despite the use of a meta-model which is believed to be time-consuming.

Conclusion

In this paper, we proposed MeLON, a meta-learning-based highly flexible update strategy for online recommender systems. MeLON provides learning rates *adaptively* to each parameter-interaction pair to help recommender systems be aligned with up-to-date user interests. To this end, MeLON first represents the meaning of a user-item interaction and the role of a parameter using a GAT and a neural mapper. Then, the adaptation layer exploits the two representations to determine the optimal learning rate. Extensive experiments were conducted using three real-world online service datasets, and the results confirmed the higher accuracy of MeLON by virtue of its *two*-directional flexibility as validated in the ablation study and theoretical analysis.

Acknowledgement

This work was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (Ministry of Science and ICT) (No. 2020R1A2B5B03095947) and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-00862, DB4DL: High-Usability and Performance In-Memory Distributed DBMS for Deep Learning).

References

Andrychowicz, M.; Denil, M.; Gomez, S.; Hoffman, M. W.; Pfau, D.; Schaul, T.; Shillingford, B.; and De Freitas, N. 2016. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, 3981–3989.

Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *PAMI*, 35(8): 1798–1828.

Davidson, J.; Liebald, B.; Liu, J.; Nandy, P.; Van Vleet, T.; Gargi, U.; Gupta, S.; He, Y.; Lambert, M.; Livingston, B.; et al. 2010. The YouTube video recommendation system. In *RecSys*, 293–296.

Du, Z.; Wang, X.; Yang, H.; Zhou, J.; and Tang, J. 2019. Sequential scenario-specific meta learner for online recommendation. In *KDD*, 2895–2904.

Eckart, C.; and Young, G. 1936. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3): 211–218.

Finn, C.; Rajeswaran, A.; Kakade, S.; and Levine, S. 2019. Online meta-learning. In *ICML*, 1920–1930.

Gulla, J. A.; Zhang, L.; Liu, P.; Özgöbek, Ö.; and Su, X. 2017. The adressa dataset for news recommendation. In *WI*, 1042–1048.

He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T.-S. 2017. Neural collaborative filtering. In *TheWebConf*, 173–182.

He, X.; Zhang, H.; Kan, M.-Y.; and Chua, T.-S. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*, 549–558.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation*, 9(8): 1735–1780.

Koren, Y. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, 426–434.

Koren, Y.; Bell, R.; and Volinsky, C. 2009. Matrix factorization techniques for recommender systems. *Computer*, 42(8): 30–37.

Li, Z.; Zhou, F.; Chen, F.; and Li, H. 2017. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*.

Linden, G.; Smith, B.; and York, J. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing*, 7(1): 76–80.

Ni, J.; Li, J.; and McAuley, J. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP-IJCNLP*, 188–197.

Ravi, S.; and Larochelle, H. 2017. Optimization as a model for few-shot learning. In *ICLR*.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*, 452–461.

Ruder, S. 2016. An overview of gradient descent optimization algorithms. *arXiv:1609.04747*.

Shu, J.; Xie, Q.; Yi, L.; Zhao, Q.; Zhou, S.; Xu, Z.; and Meng, D. 2019. Meta-Weight-Net: Learning an explicit mapping for sample weighting. In *NeurIPS*, 1919–1930.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In *ICLR*.

Zhang, Y.; Feng, F.; Wang, C.; He, X.; Wang, M.; Li, Y.; and Zhang, Y. 2020. How to retrain recommender system? A sequential meta-learning method. In *SIGIR*, 1479–1488.

Supplementary Material for Paper 2570: Meta-Learning for Online Update of Recommender Systems

A. Online Recommender Training

While an online recommender can be trained on new user-item interactions with adaptive learning rates by the meta-model MeLON, the optimality condition varies with time. Therefore, MeLON should be continuously updated along with the recommender to avoid being stale. To this end, as shown in Figure 6, for every incoming mini-batch $\mathcal{B}_t$, we first conduct two steps to train the *meta-model* ϕ^{2D} before updating the *recommender model* $\mathbf{\Theta}$, following the common update procedure of meta-learning (Ren et al. 2018; Shu et al. 2019). Note that Figure 6 is made more detailed by clarifying the two steps for ϕ^{2D}, compared with Figure 3 (in the main paper).

1. *Recommender model preliminary update*: For the users and items in the new mini-batch $\mathcal{B}_t$ at each iteration, we first derive their last interactions $\mathcal{B}_t^{last}$ before the current interaction. Then, using the *current* meta-model ϕ_t^{2D}, the parameter $\mathbf{\Theta_t}$ of the recommender model is updated on the latest interactions $\mathcal{B}_t^{last}$ to create a model with $\tilde{\mathbf{\Theta}}$ by

$$\tilde{\Theta} = \Theta_t - \nabla_{\Theta_t} \sum_{x \in \mathcal{B}_t^{last}} \mathcal{L}_{\Theta_t}(x) \cdot \phi_t^{2D}\Big(x, \mathcal{L}_{\Theta_t}(x), \Theta_t\Big). \quad (14)$$

2. *Meta-model update*: Because $\tilde{\Theta}$ obtained by Eq. (14) is widely known as an inspection on the efficacy of the current meta-model (Ren et al. 2018; Shu et al. 2019), the feedback from $\tilde{\Theta}$ is exploited to update the meta-model on the incoming mini-batch $\mathcal{B}_t$ by

$$\phi_{t+1}^{2D} = \phi_t^{2D} - \eta \nabla_{\phi_t^{2D}} \sum_{x \in \mathcal{B}_t} \frac{1}{n} \mathcal{L}_{\tilde{\Theta}}(x), \quad (15)$$

   where η is a learning rate for meta-model.

3. *Recommender model update*: Finally, the parameter $\mathbf{\Theta}_t$ of the recommender model is updated using the *updated* meta-model ϕ_{t+1}^{1D} on the mini-batch $\mathcal{B}_t$ by

$$\Theta_{t+1} = \Theta_t - \nabla_{\Theta_t} \sum_{x \in \mathcal{B}_t} \mathcal{L}_{\Theta_t}(x) \cdot \phi_{t+1}^{2D}\Big(x, \mathcal{L}_{\Theta_t}(x), \Theta_t\Big). \quad (16)$$

Note that MeLON *selectively* performs the update of recommender parameters involved in the recommender's computation for each interaction. Therefore, the required update time for MeLON is comparable to other update strategies, such as SML and S^{2}Meta, as empirically confirmed in the evaluation results.

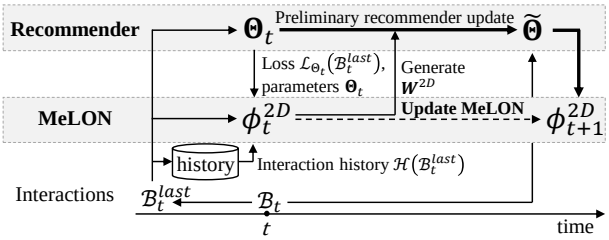


Figure 6: MeLON update procedure. For the users and items in every new mini-batch $\mathcal{B}_t$, we retrieve their last interaction $\mathcal{B}_t^{last}$ to conduct a preliminary update of the recommender model with MeLON. Then, we update MeLON based on the loss of the updated model $\tilde{\Theta}$ on the new mini-batch $\mathcal{B}_t$.

Algorithm 1: Online Training via MeLON

INPUT: Θ_t: recommender model, ϕ_t^{2D}: meta-model

1: $t \leftarrow 1$; $\Theta_t, \phi_t^{2D} \leftarrow$ Load pretrained models;
2: **while** user-item interactions are coming **do**
3: $\quad \mathcal{B}_t \leftarrow$ Get current mini-batch;
4: $\quad$ /* **Meta-model update** */
5: $\quad$ /* (1) Preliminary update by Eq. (14) */
6: $\quad \tilde{\Theta} = \Theta_t - \nabla_{\Theta_t} \sum_{x \in \mathcal{B}_t^{last}} \mathcal{L}_{\Theta_t}(x) \cdot \phi_t^{2D}\big(x, \mathcal{L}_{\Theta_t}(x), \Theta_t\big)$;
7: $\quad$ /* (2) Meta-model update by Eq. (15) */
8: $\quad \phi_{t+1}^{2D} = \phi_t^{2D} - \eta \nabla_{\phi_t^{2D}} \sum_{x \in \mathcal{B}_t} \frac{1}{n} \mathcal{L}_{\tilde{\Theta}}(x)$;
9: $\quad$ /* **Recommender update** */
10: $\quad$ /* (3) Recommender model update by Eq. (16) */
11: $\quad \Theta_{t+1} = \Theta_t - \nabla_{\Theta_t} \sum_{x \in \mathcal{B}_t} \mathcal{L}_{\Theta_t}(x) \cdot \phi_{t+1}^{2D}\big(x, \mathcal{L}_{\Theta_t}(x), \Theta_t\big)$;
12: $\quad t \leftarrow t + 1$;

The online training procedure of MeLON is described in Algorithm 1. When a recommender is deployed online, the algorithm conducts the three steps for every new incoming mini-batch of user-item interactions: (1) a preliminary update of the recommender model (Lines 5–6) on the last interactions of the users and items in the current mini-batch, (2) an update of the meta-model on a new mini-batch (Lines 7–8), and (3) an update of the recommender model on the new mini-batch (Lines 10–11). We additionally learn a forgetting rate for the current parameter to help quick adaptation by forgetting previous outdated information (Ravi and Larochelle 2017). The above procedure repeats for every incoming mini-batch during online service.

Before a recommender is deployed online, both the recommender and the meta-model are typically pre-trained on

the past user-item interactions in an offline manner. Differently from the online training, we first randomly sample a mini-batch $\mathcal{B}$ of user-item interactions to derive the interactions $\mathcal{B}^{last}$. Then, in each iteration, the recommender and the meta-model are updated in the same way as in the online learning. The model is trained for a fixed number of epochs, 100 in our experiments. Once the offline training completes, we can deploy the recommender and the meta-model in the online recommendation environment.

B. Details of Experiment Settings

Four reproducibility, the source code of MeLON as well as the datasets are provided as the supplementary material.

Datasets

The explicit user ratings in the Yelp dataset and three Amazon datasets are converted into implicit ones, following the relevant researches (Koren 2008; He et al. 2017); that is, if a user rated an item, then the rating is considered as a positive user-item interaction. For prequential evaluation on online recommendation scenarios, we follow a commonly-used approach (He et al. 2016); we sort the interactions in the dataset in chronological order, and divide them into three parts—offline pre-training data, online validation data, and online test data. Online validation data is exploited to search the hyperparameter setting of the recommenders and update strategies and takes up 10% of test data. Because user-item interactions are very sparse, we preprocess the datasets, following the previous approaches (He et al. 2017; Zhang et al. 2020); for all datasets, users and items involved with less than 20 interactions are filtered out.

Table 5 summarizes the profiles of the five datasets used in the experiments, where the details are as follows.

Adressa news dataset (Gulla et al. 2017) contains user interactions with news articles for one week. We use the first 95% of data as offline pre-training data, the next 0.5% as online validation data, and the last 4.5% as online test data.

Amazon review dataset (Ni, Li, and McAuley 2019) contains user reviews for the products purchased in Amazon. Among various categories, we adopt three frequently-used categories, *Book* (Wang et al. 2019), *Electronics* (Zhou et al. 2018), and *Grocery and Gourmet Food* (Wang et al. 2020), which vary in the size of interactions and the number of users and items. Because there exists almost no overlap among the categories, we perform evaluation on each category and report the average. Due to the difference in size, we apply different data split ratios for each category.

- Book: 95% (pre-training): 0.5% (validation): 4.5% (test)
- Electronics: 90% (pre-training):1% (validation):9% (test)
- Grocery: 80% (pre-training): 2% (validation): 18% (test)

Yelp review dataset[4] contains user reviews for venues, such as bars, cafes, and restaurants. We use the first 95% of data as offline pre-training data, the next 0.5% as online validation data, and the last 4.5% as online test data.

[4]https://www.kaggle.com/yelp-dataset/yelp-dataset

Table 5: Summary of the five real-world datasets. A cold user refers to the users who do not exist in the pre-training data but in the online test data.

Dataset	Users	Items	Interactions	Cold user
Adressa	29,589	1,457	1,191,114	0%
Amazon (Book)	80,464	98,663	3,357,109	18%
Amazon (Electronics)	9,316	17,935	238,458	1%
Amazon (Grocery)	1,233	1,433	29,782	1%
Yelp	60,543	74,249	2,880,520	0%

Recommender Baseline Models

For online recommenders, we use two famous personalized recommender algorithms: BPR (Koren, Bell, and Volinsky 2009; Rendle et al. 2009) and NCF (He et al. 2017).

- BPR: To handle implicit feedback, the Bayesian personalized ranking (BPR) uses the identifiers of a user and an item to estimate the user's interest on the item by multiplying the user embedding vector $\mathbf{e}_u$ and the item embedding vector $\mathbf{e}_i$.
- NCF: Neural collaborative filtering (NCF) maintains a generalized BPR and a multi-layer perceptron that have user and item vectors respectively. The results of these two components are later fused by a neural layer to predict a user's interest on an item.

To train these recommender algorithms based on implicit feedback data, we employ a ranking loss (Rendle et al. 2009); for a positive item in a user-item interaction, we randomly sample another negative item that the user has not interacted before, and train a recommender algorithm to prioritize the positive item over the negative item.

Configuration

For fair comparison, we follow the optimal hyperparameter settings of the baselines as reported in the original papers, and optimize uncharted ones using HR@5 on validation data. All the experiments are performed with a batch size 256 and trained for 100 epochs. The number of updates on the default update strategy is fixed to be 1 to align with other compared strategies. The experiments are repeated 5 times varying random seeds, and we report the average as well as the standard error. For the graph attention in the first component of MeLON, we randomly sample 10 neighbors per target user or item. Besides, for the MLP which learns the parameter roles, the number of hidden layers (L) is set to be 2. To optimize a recommender under the default and sample reweighting strategies, we use Adam (Kingma and Ba 2015) with a learning rate $\eta = 0.001$ and a weight decay 0.001. Note that a recommender trained with the meta-optimization strategies is optimized by a meta-model, while the meta-model is optimized with Adam during the meta-update in Eqs. (14) and (15). Our implementation is written in PyTorch, and the experiments were conducted on Nvidia Titan RTX.

References

Gulla, J. A.; Zhang, L.; Liu, P.; Özgöbek, Ö.; and Su, X. 2017. The adressa dataset for news recommendation. In *WI*, 1042–1048.

He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T.-S. 2017. Neural collaborative filtering. In *TheWebConf*, 173–182.

He, X.; Zhang, H.; Kan, M.-Y.; and Chua, T.-S. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*, 549–558.

Kingma, D. P.; and Ba, J. 2015. Adam: A method for stochastic optimization. In *ICLR*.

Koren, Y. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, 426–434.

Koren, Y.; Bell, R.; and Volinsky, C. 2009. Matrix factorization techniques for recommender systems. *Computer*, 42(8): 30–37.

Ni, J.; Li, J.; and McAuley, J. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP-IJCNLP*, 188–197.

Ravi, S.; and Larochelle, H. 2017. Optimization as a model for few-shot learning. In *ICLR*.

Ren, M.; Zeng, W.; Yang, B.; and Urtasun, R. 2018. Learning to reweight examples for robust deep learning. In *ICML*, 4334–4343.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*, 452–461.

Shu, J.; Xie, Q.; Yi, L.; Zhao, Q.; Zhou, S.; Xu, Z.; and Meng, D. 2019. Meta-Weight-Net: Learning an explicit mapping for sample weighting. In *NeurIPS*, 1919–1930.

Wang, C.; Zhang, M.; Ma, W.; Liu, Y.; and Ma, S. 2020. Make it a chorus: knowledge-and time-aware item modeling for sequential recommendation. In *SIGIR*, 109–118.

Wang, X.; He, X.; Wang, M.; Feng, F.; and Chua, T.-S. 2019. Neural graph collaborative filtering. In *SIGIR*, 165–174.

Zhang, Y.; Feng, F.; Wang, C.; He, X.; Wang, M.; Li, Y.; and Zhang, Y. 2020. How to retrain recommender system? A sequential meta-learning method. In *SIGIR*, 1479–1488.

Zhou, G.; Zhu, X.; Song, C.; Fan, Y.; Zhu, H.; Ma, X.; Yan, Y.; Jin, J.; Li, H.; and Gai, K. 2018. Deep interest network for click-through rate prediction. In *KDD*, 1059–1068.