Task Relation-aware Continual User Representation Learning

Sein Kim*
KAIST
Republic of Korea
rlatpdlsgns@kaist.ac.kr

Namkyeong Lee*
KAIST
Republic of Korea
namkyeong96@kaist.ac.kr

Donghyun Kim
NAVER Corporation
Republic of Korea
amandus.kim@navercorp.com

Minchul Yang
NAVER Corporation
Republic of Korea
minchul.yang@navercorp.com

Chanyoung Park†
KAIST
Republic of Korea
cy.park@kaist.ac.kr

ABSTRACT

User modeling, which learns to represent users into a low-dimensional representation space based on their past behaviors, got a surge of interest from the industry for providing personalized services to users. Previous efforts in user modeling mainly focus on learning a task-specific user representation that is designed for a single task. However, since learning task-specific user representations for every task is infeasible, recent studies introduce the concept of universal user representation, which is a more generalized representation of a user that is relevant to a variety of tasks. Despite their effectiveness, existing approaches for learning universal user representations are impractical in real-world applications due to the data requirement, catastrophic forgetting and the limited learning capability for continually added tasks. In this paper, we propose a novel continual user representation learning method, called TERACON, whose learning capability is not limited as the number of learned tasks increases while capturing the relationship between the tasks. The main idea is to introduce an embedding for each task, i.e., *task embedding*, which is utilized to generate task-specific soft masks that not only allow the entire model parameters to be updated until the end of training sequence, but also facilitate the relationship between the tasks to be captured. Moreover, we introduce a novel knowledge retention module with pseudo-labeling strategy that successfully alleviates the long-standing problem of continual learning, i.e., catastrophic forgetting. Extensive experiments on public and proprietary real-world datasets demonstrate the superiority and practicality of TERACON. Our code is available at https://github.com/Sein-Kim/TERACON.

CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**.

KEYWORDS

Continual learning, Universal User Representation, Recommender System

ACM Reference Format:
Sein Kim, Namkyeong Lee, Donghyun Kim, Minchul Yang, and Chanyoung Park. 2023. Task Relation-aware Continual User Representation Learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3580305.3599516

1 INTRODUCTION

To identify relevant information from the overloaded data, machine learning (ML) has become a crucial tool for social media and e-commerce platforms, boosting not only user experience but also business revenue [3]. Among the various ML techniques, user modeling (UM) got a surge of interest from the industry due to its ability in discovering user's latent interests, thereby enabling personalized services for a variety of users [9, 12, 14, 16, 20, 43]. The key idea in UM is to represent users into a low-dimensional representation space based on their past behaviors, which inherently contain rich and diverse users' interests.

Despite the significant progress in UM, most of the previous efforts have been concentrated on learning task-specific user representations, which are specifically designed for a single task (e.g., click-through rate prediction, user profiling) [7, 43, 48], limiting its applicability to real-world applications. For example, companies like Amazon and Alibaba offer various services to users in a single platform, all of which require service-specific UM to enhance user satisfaction and improve business revenue [2, 33, 35]. However, learning service-specific user representations for every service is impractical in reality due to expensive memory/financial costs. To overcome the limitations of task-specific user representations, the concept of a universal user representation has recently emerged as a solution. The key idea is to learn a generalized representation of a user that better captures the user's underlying characteristics, which are relevant for not only single task but also across a variety of tasks [6, 26, 42, 45].

Existing studies mainly adopt multi-task learning (MTL) [4, 30] to learn universal user representations. MTL aims to learn a generalized representation of users by simultaneously training a single model for each user on multiple tasks [22, 27]. However, MTL has inherent limitations as it requires all tasks and their associated data to be available in advance, which rarely holds in real-world

[*] Work done while the authors were interning at NAVER Corporation.
[†] Corresponding author.

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599516

situations. In other words, when introducing a new service, MTL-models need to be retrained on both the new data and all the data from previous tasks. Meanwhile, Transfer learning (TL) offers an alternative approach to address this issue. More precisely, given a pair of tasks, i.e., source task with abundant data and target task with insufficient data, TL aims to improve the performance on the target task by transferring the knowledge obtained from the source task, which is different but positively related to the target task [39, 42, 47]. Specifically, PeterRec [42] boosts the model performance on various target tasks by pre-training the model on the user-item interaction in a source task in a self-supervised manner. However, TL is only applicable when a pair of tasks is given, whereas online platforms usually contain multiple target tasks. For this reason, even if multiple target tasks are positively related, i.e., offer useful information to each other, knowledge transfer can only happen between a source-target pair. Moreover, the model easily suffers from catastrophic forgetting, that is, the model performance on the source task severely deteriorates after training the model on the target task [25, 28].

Recently, continual learning (CL) has shown great success in learning a single model on a stream of multiple tasks while retaining the knowledge from the tasks observed in the past [1, 5, 19], which alleviates the shortcomings of MTL and TL mentioned above. More precisely, CL is similar to MTL in that the model learns from multiple tasks, but CL does not require the simultaneous availability of training data across multiple target tasks. Besides, CL is similar to TL in that the model can transfer knowledge from previous tasks to new tasks. However, CL also retains the knowledge from previous tasks, and some CL models capture the relationship between multiple target tasks [13, 18]. CONURE [45] is the first work to adopt the CL framework to recommender systems, and it learns a universal user representation based on a series of multiple tasks. The main idea is to train the model on a task using only a portion of the model parameters, which are then fixed when being trained on the next task where another portion of the remaining model parameters is used. This enables the model to retain the knowledge obtained in the past, i.e., avoid catastrophic forgetting, and such an approach is called parameter isolation.

Despite its effectiveness, CONURE fails to learn from the continuously incoming sequence of tasks due to the inherent limitation of the parameter isolation approach, which restricts the modification of model parameters that are used in previous tasks. While this approach prevents catastrophic forgetting, it limits the model's capability to learn subsequent tasks due to the gradual reduction of available learnable parameters as more tasks are added. Another drawback of CONURE is that it does not consider the relationship between tasks. For example, the knowledge obtained from predicting a user's age could enhance item purchase prediction, since certain items may be popular among users of a specific age, whereas the age prediction task would not be helpful for predicting the gender of a user. Hence, capturing the relationship between tasks encourages a positive transfer between positively related tasks, while preventing a negative transfer between negatively related tasks [13, 46], and thus it is crucial for an effective training of a new task.

To this end, we propose a novel Task Embedding-guided Relation-Aware CONtinual user representation learner, named TERACON, whose learning capability is not limited regardless of the number of continually added tasks, while capturing the relationship between tasks. The main idea is to introduce an embedding for each task, i.e., *task embedding*, which is utilized to generate a task-specific soft mask that not only allows the entire model parameters to be updated until the end of the training sequence thereby retaining the learning capability of the model, but also facilitates the relationship between the tasks to be captured. Moreover, to prevent catastrophic forgetting, we propose a knowledge retention module that transfers the knowledge of the current model regarding previous tasks to help train the current model itself. Lastly, we propose a sampling strategy to make the training more efficient. Our extensive experiments on two real-world datasets and a proprietary dataset demonstrate that TERACON outperforms the state-of-the-art continual user representation learning methods.

Our contributions are summarized as follows:

- In this work, we propose a novel continual user representation learning method, called TERACON, that 1) retains the learning capability until the end of the training sequence, 2) captures the relationship between tasks. and, 3) prevents catastrophic forgetting through a novel knowledge retention module with pseudo-labeling.
- For an efficient training of TERACON, we propose a relation-aware user sampling strategy that samples users from each task considering the relationship between tasks.
- Extensive experiments on two public and one proprietary datasets demonstrate the superiority of TERACON compared to the recent state-of-the-art methods. A further appeal of TERACON is its robustness in real-world scenarios, verifying the possibility of adopting TERACON on various web platforms.

2 RELATED WORK

2.1 Universal User Representation

User modeling (UM) refers to the process of obtaining the user profile, which is a conceptual understanding of the user for personalized recommender systems. The key idea of UM is to learn the representation for each user by leveraging the user's interacted items or the features of the items, and the obtained representations are used for a wide range of applications such as response prediction and recommendation [21]. Early methods aim to learn a user representation via matrix factorization, whose matrix consists of user-item interaction history, assuming that a user can be represented based on the interacted items [29, 31]. Along with the recent advances of deep neural networks, neural network based user modeling got a surge of interest from researchers, including factorization-based approaches [7, 8], recurrent neural network based approaches [9], and graph-based approaches [36, 38]. However, these UM approaches focus on task-specific user representations, which may not represent the generalized interest of users.

Recently, several studies have adopted multi-task learning (MTL) or transfer learning (TL) to move from task-specific user representations to universal user representations. MTL aims to simultaneously learn multiple tasks with a single shared representation for each user. For example, DUPN [27] introduces an attention mechanism to integrate all content, behavior, and other relevant information from multiple tasks to generate a user representation. ESM^2 [37]

utilizes the conditional probability of user behavior graph (e.g., impression ⟶ click ⟶ purchase) to explicitly express the task relation for multi-task learning. On the other hand, TL exploits the knowledge (i.e., user representation) gained in a source task, which is different but positively related to the target task, to improve the performance of target task [10, 49]. For example, DARec [44] models the rating pattern in the source task and transfers the knowledge to the target task for the cross-domain recommendation. Moreover, PeterRec [42] boosts the model performance on various target tasks by pre-training the model with sequential user-item interactions in a source task in a self-supervised manner.

Despite the success of MTL and TL in certain applications, they also have limitations when it comes to real-world scenarios. More precisely, MTL requires all the tasks and their associated data to be available in advance, which is problematic when a new service is to be launched as the model should be retrained with all the data. Moreover, TL can be done with only two tasks, i.e., the source task and the target task, which is impractical for real-world online platforms that contain multiple target tasks. Distinguished from MTL and TL, we propose to continually learn universal user representations, which is more practical in reality.

2.2 Continual Learning

It is widely known that neural networks tend to forget the knowledge of previously learned tasks when they are trained on a new task, and this phenomenon is called catastrophic forgetting [25, 28]. Continual learning aims to prevent such catastrophic forgetting issues during the training of a stream of various tasks. Recently proposed continual learning approaches can be divided into three categories, i.e., replay-based, architecture-based, and parameter regularization-based approaches. **Replay-based approaches** prevent catastrophic forgetting by storing a subset of data of previous tasks into a replay buffer, and use them when training on the next task. Therefore, selecting the subset of data that best represents the data distribution in the previous tasks is the key to the success of replay-based approaches [17, 34, 40]. **Architecture-based approaches** prevent catastrophic forgetting by expanding the capacity of the model architecture when it is insufficient to train on new tasks [11, 41]. **Regularization-based approaches** regularize the model parameters to minimize catastrophic forgetting. The key idea is to restrict significant changes in the model parameters during training on new tasks by regularizing the model parameters that were important in previous tasks [15, 23]. Recently, PackNet [24] leverages binary masks to restrict the update of the parameters that were shown to be important in previous tasks.

Inspired by the success of PackNet, CONURE [45] proposes to learn the universal user representation by keeping the parameters that were crucial in previous tasks. Specifically, CONURE continuously learns the sequence of tasks by iteratively removing less important parameters and saving the crucial parameters for each task, which is also called network pruning. By freezing the crucial parameters in previous tasks, CONURE is able to learn new tasks while retaining the knowledge obtained from the previous tasks, and such an approach is called parameter isolation. However, such parameter isolation-based methods restricts the modification of model parameters that are used in previous tasks, and thus the model's learning capacity is gradually reduced as new task are introduced due to the lack of available learnable parameters. Moreover, the knowledge obtained from the new tasks cannot be transferred to the previous tasks, i.e., backward transfer, which can be also helpful for previous tasks.

3 PRELIMINARIES

In this section, we introduce a formal definition of the problem including the notations and the task description (Sec. 3.1). Then, we briefly introduce NextitNet [43], which is used as the backbone network of our model (Sec. 3.2), and how the backbone network is trained in the sequence of tasks $\mathcal{T}$ (Sec. 3.3).

3.1 Problem Formulation

Notations. Let $\mathcal{T} = \{T_1, T_2, \cdots, T_M\}$ denote the set of consecutive tasks, which can be also represented as $T_{1:M}$. Let $\mathcal{U} = \{u_1, u_2, \cdots, u_N\}$ denote the set of users. Each user $u_l \in \mathcal{U}$ is represented by his/her behavior sequence $\mathbf{x}^{u_l} = \left\{x_1^{u_l}, x_2^{u_l}, \cdots, x_n^{u_l}\right\}$, where $x_t^{u_l} \in \mathcal{I}$ is the t-th interaction of u_l and $\mathcal{I}$ is the set of items. For each task T_i, only a subset of users $\mathcal{U}^{T_i} = \{u_1, u_2, \cdots, u_{|\mathcal{U}^{T_i}|}\}$ exist, i.e., $\mathcal{U}^{T_i} \subset \mathcal{U}$, and the set of users in task T_i is associated with the set of labels $\mathrm{Y}^{T_i} = \{y_{u_1}^{T_i}, \ldots, y_{u_{|\mathcal{U}^{T_i}|}}^{T_i}\}$, where $y_{u_l}^{T_i}$ denotes the label of user u_l in task T_i (e.g., purchased item, gender, and age) of user u_l. We use $\mathcal{Y}^{T_i}$ to denote the set of unique labels in Y^{T_i}, and $\mathbf{y}_{u_l}^{T_i} \in \{0, 1\}^{|\mathcal{Y}^{T_i}|}$ denotes the one-hot transformation of $y_{u_l}^{T_i}$. Note that the first task T_1 contains all the users in the dataset, i.e., $\mathcal{U}^{T_1} = \mathcal{U}$.

Task: Continual User Representation Learning. Assume that we are given the set of consecutive tasks $\mathcal{T} = \{T_1, T_2, \cdots, T_M\}$, where each task T_i is associated with the set of users $\mathcal{U}^{T_i}$ and the set of labels Y^{T_i}. Our goal is to train a single model $\mathcal{M}$ on each task $T_i \in \mathcal{T}$ one by one in a sequential manner to predict the label of each user $u_l \in \mathcal{U}^{T_i}$, i.e., $\mathbf{y}_{u_l}^{T_i} = G^{T_i}(\mathcal{M}(\mathbf{x}^{u_l}))$, where $\mathcal{M}$ is the backbone encoder network that generates universal user representations, and G^{T_i} is a task-specific classifier for task T_i. After training the entire sequence of tasks from T_1 to T_M, the single model $\mathcal{M}$ is used to serve all tasks in $\mathcal{T}$.

3.2 Model Backbone: TCN

Following a previous work [45], we adopt temporal convolutional network (TCN) [43] as the backbone network of TERACON, although our framework is network-agnostic. TCN learns the representation of a user u_l based on the sequence of the user's interacted items, i.e., $\mathbf{x}^{u_l} = \left\{x_1^{u_l}, x_2^{u_l}, \cdots, x_n^{u_l}\right\}$. More precisely, given an initial embedding matrix of $\mathbf{x}^{u_l}$, i.e., $\mathbf{E}_0^{u_l} \in \mathbb{R}^{n \times f}$ where f is the emebdding size, we pass it through a TCN, which is composed of a stack of residual blocks, each containing two temporal convolutional layers and normalization layers. The k-th residual block is given as follows:

$$\mathbf{E}_k^{u_l} = F_k(\mathbf{E}_{k-1}^{u_l}) + \mathbf{E}_{k-1}^{u_l} = R_k(\mathbf{E}_{k-1}^{u_l}) \tag{1}$$

where $k = 1, \ldots, K$, $\mathbf{E}_{k-1}^{u_l}$ and $\mathbf{E}_k^{u_l}$ are the input and output of the residual block in layer k, respectively, F_k is a residual mapping in layer k, which is composed of two layers of convolution, layer normalization and ReLU activation, and R_k is the residual block in layer k.

3.3 Task Incremental Learning

During training, our backbone network sequentially learns the tasks in $\mathcal{T}$ one by one, i.e., $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_M$. Following [45], the same user behavior sequence $\mathbf{x}^{u_l}$ is used as the model input for all tasks, whereas the target domain $y_{u_l}^{T_i}$ varies as the task varies, e.g., item purchase prediction for T_1, user age prediction for T_2, and user gender prediction for T_3, etc.

Training the First Task (T_1). In the first task T_1, we generate base user representations, which will be used in the following tasks, based on the behavior sequence of user $u_l \in \mathcal{U}^{T_1}$, i.e., $\mathbf{x}^{u_l}$. To do so, we train our backbone network in a self-supervised manner to autoregressively predict the next item in the user behavior sequence $\mathbf{x}^{u}$ as follows:

$$p(\mathbf{x}^{u_l};\Theta) = \prod_{j=1}^{n-1} p(x_{j+1}^{u_l} | x_1^{u_l}, \cdots, x_j^{u_l};\Theta), \tag{2}$$

where $p(x_{j+1}^{u_l} | x_1^{u_l}, \cdots, x_j^{u_l};\Theta)$ indicates probability of the $(j+1)$-th interaction with user u_l conditioned on the user's past interaction history $\{x_1^{u_l}, \cdots, x_j^{u_l}\}$, and Θ is the set of parameters of TCN. By training TCN to maximize the above joint probability distribution of the user behavior sequence $\mathbf{x}^{u_l}$, we obtain the base user representation for user u_l, which can be transferred to various tasks $T_{2:i}$.

Training of T_i ($i > 1$). After training the first task T_1, we continually learn the subsequent tasks $T_{2:i}$ with our backbone network whose parameters are pre-trained based on T_1. Given the behavior sequence of user u_l, i.e., $\mathbf{x}^{u_l} = \{x_1^{u_l}, x_2^{u_l}, \cdots, x_n^{u_l}\}$, our backbone network autoregressively outputs the embeddings, i.e., $\mathbf{E}_K^{u_l} \in \mathbb{R}^{n \times f}$, for predicting the next item. More precisely, the j-th row of $\mathbf{E}_K^{u_l}$ is the output of the network when the behavior sequence $\{x_1^{u_l}, x_2^{u_l}, \cdots, x_{j-1}^{u_l}\}$ is given. Hence, we use the last row of $\mathbf{E}_K^{u_l}$ as the input to the task-specific classifier of T_i to obtain the label (e.g., predicting the next purchased item, gender, age, etc) for user u_l as follows:

$$\hat{\mathbf{y}}_{u_l}^{T_i} = \mathbf{E}_K^{u_l}[-1,:]\mathbf{W}^{T_i} + \mathbf{b}^{T_i} = G^{T_i}(\mathbf{E}_K^{u_l}) \tag{3}$$

where $\hat{\mathbf{y}}_{u_l}^{T_i} \in \mathbb{R}^{|\mathcal{Y}^{T_i}|}$ is the prediction of labels of u_l in task T_i, $\mathbf{E}_K^{u_l}[-1,:] \in \mathbb{R}^f$ is the last row of $\mathbf{E}_K^{u_l}$, $\mathbf{W}^{T_i} \in \mathbb{R}^{f \times |\mathcal{Y}^{T_i}|}$ and $\mathbf{b}^{T_i} \in \mathbb{R}^{|\mathcal{Y}^{T_i}|}$ denote the task-specific projection matrix and bias term of fully-connected-layer for T_i, respectively, and G^{T_i} is a simplified notation of the task-specific classifier of T_i.

3.4 Parameter Isolation

However, as shown in [45] the model that is naively trained with various tasks in a sequential manner suffers from catastrophic forgetting [25, 28]. That is, the model performance on the previous tasks $T_{1:(i-1)}$ drastically deteriorates when learning on a new task T_i. To this end, CONURE [45] proposes to alleviate catastrophic forgetting by adopting the parameter isolation approach, which freezes a portion of model parameters that were crucial in learning previous tasks. Specifically, CONURE learns the new task T_i with the following model parameters:

$$\mathbf{Z}_k^{T_i} = \text{stop\_grad}\left[\hat{\mathbf{Z}}_k^{T_{i-1}} \odot \sum_{t=1}^{i-1} \mathbf{M}_k^{T_t}\right] + \hat{\mathbf{Z}}_k^{T_{i-1}} \odot (\mathbf{1}_k - \sum_{t=1}^{i-1} \mathbf{M}_k^{T_t}) \tag{4}$$

where $\odot$ is element-wise product operator, stop_grad is an operator that prevents back propagation of gradients, $\hat{\mathbf{Z}}_k^{T_{i-1}} \in \mathbb{R}^{a \times b}$ is the parameter of the k-th layer of TCN that is used for learning T_{i-1}, where $a \times b$ is the shape of the parameter[1], and $\mathbf{1}_k \in \mathbb{R}^{a \times b}$ is a matrix whose elements are all ones. Moreover, $\mathbf{M}_k^{T_t} \in \mathbb{R}^{a \times b}$ indicates a task-specific binary mask in layer k for protecting model parameters that were considered to be significant in previous tasks $T_{1:(i-1)}$, which is obtained as follows:

$$\mathbf{M}_k^{T_i} = \begin{cases} 1 & \text{if } |\mathbf{Z}_k^{T_i}| \odot (\mathbf{1}_k - \sum_{t=1}^{i-1} \mathbf{M}_k^{T_t}) > \delta, \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where $|\cdot|$ denotes the symbol of element-wise absolute value, and δ is a threshold hyper-parameter for selecting the significant parameters, which is determined in advance according to the amount of parameters to be frozen in each task. In other words, Equation 5 aims to keep an element of the parameter $\mathbf{Z}_k^{T_i}$ frozen, if its absolute value is greater than a threshold δ. In the inference phase, given a T_t, the model loads the masks that are stored to generate the representation, i.e., $\tilde{\mathbf{Z}}_k^{T_t} = \hat{\mathbf{Z}}_k^{T_t} \odot \sum_{j=1}^{t} \mathbf{M}_k^{T_j}$. In the inference phase, since CONURE utilizes the parameters that were kept frozen after learning each task, it prevents catastrophic forgetting.

Limitation of CONURE. However, we argue that CONURE has the following drawbacks: **1)** By adopting the parameter isolation approach, the number of available learnable parameters is gradually reduced as more tasks are continually added, which limits the model's capability in learning subsequent tasks even making it impossible to learn new tasks after using up all the remaining parameters. Moreover, once the parameters are frozen, the previous tasks cannot benefit from the subsequent tasks whose knowledge can also be beneficial to previous tasks, i.e., positive backward transfer. **2)** Besides the limitations incurred by the parameter isolation, CONURE does not consider the relationship between tasks, which however is beneficial in two perspectives. That is, it encourages the knowledge learned from positively related tasks to be better transferred between the one another, and at the same time, it prevents the knowledge learned from the negatively related task from disturbing one another. Therefore, we need a model that learns universal user representations that a) retains the learning capability until the end of the training sequence, while b) considering the relationship between tasks.

4 PROPOSED METHOD: TERACON

In this section, we introduce a novel universal user representation learning model, called TERACON, whose key component is the *task embedding*, which not only generates a task-specific soft mask, but also facilitates the relationship between the tasks to be captured.

4.1 Learning Task-specific Mask via Task Embedding

We begin by describing how the task embedding and the task-specific mask are defined. We use $\mathbf{e}_k^{T_i} \in \mathbb{R}^f$ and $\mathbf{m}_k^{T_i} \in \mathbb{R}^f$ to denote the task embedding and the task-specific mask of task T_i in layer k,

[1] Although the convolution layer in TCN has 3D filters, we reshape it to $a \times b$ for simplicity of explanation.

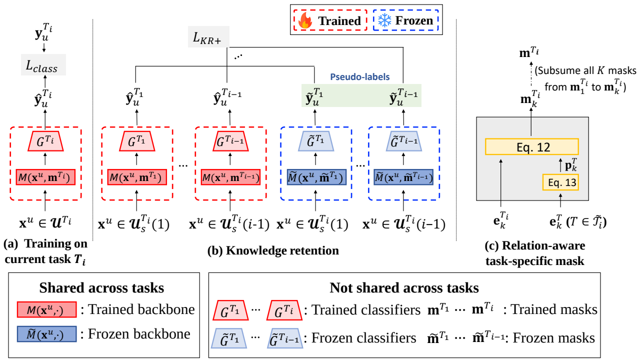
Trained
Frozen
$\mathbf{y}_u^{T_i}$
L_{class}
$\hat{\mathbf{y}}_u^{T_i}$
G^{T_i}
$M(\mathbf{x}^u, \mathbf{m}^{T_i})$
$\mathbf{x}^u \in \mathcal{U}^{T_i}$
(a) Training on current task T_i
L_{KR+}
Pseudo-labels
$\hat{\mathbf{y}}_u^{T_1}$
$\hat{\mathbf{y}}_u^{T_{i-1}}$
$\tilde{\mathbf{y}}_u^{T_1}$
$\tilde{\mathbf{y}}_u^{T_{i-1}}$
G^{T_1}
$G^{T_{i-1}}$
$\tilde{G}^{T_1}$
$\tilde{G}^{T_{i-1}}$
$M(\mathbf{x}^u, \mathbf{m}^{T_1})$
$M(\mathbf{x}^u, \mathbf{m}^{T_{i-1}})$
$\tilde{M}(\mathbf{x}^u, \tilde{\mathbf{m}}^{T_1})$
$\tilde{M}(\mathbf{x}^u, \tilde{\mathbf{m}}^{T_{i-1}})$
$\mathbf{x}^u \in \mathcal{U}_s^{T_i}(1)$
$\mathbf{x}^u \in \mathcal{U}_s^{T_i}(i\text{-}1)$
$\mathbf{x}^u \in \mathcal{U}_s^{T_i}(1)$
$\mathbf{x}^u \in \mathcal{U}_s^{T_i}(i-1)$
(b) Knowledge retention
$\mathbf{m}^{T_i}$
(Subsume all K masks from $\mathbf{m}_1^{T_i}$ to $\mathbf{m}_k^{T_i}$)
$\mathbf{m}_k^{T_i}$
Eq. 12
$\mathbf{p}_k^T$
Eq. 13
$\mathbf{e}_k^{T_i}$
$\mathbf{e}_k^T$ $(T \in \tilde{\mathcal{T}}_i)$
(c) Relation-aware task-specific mask
Shared across tasks
$M(\mathbf{x}^u,\cdot)$: Trained backbone
$\tilde{M}(\mathbf{x}^u,\cdot)$: Frozen backbone
Not shared across tasks
G^{T_1} ··· G^{T_i} : Trained classifiers
$\mathbf{m}^{T_1}$ ··· $\mathbf{m}^{T_i}$: Trained masks
$\tilde{G}^{T_1}$ ··· $\tilde{G}^{T_{i-1}}$: Frozen classifiers
$\tilde{\mathbf{m}}^{T_1}$ ··· $\tilde{\mathbf{m}}^{T_{i-1}}$: Frozen masks

Figure 1: Overall model framework. (a) Training the current task T_i. (b) Knowledge retention from previous tasks $T_{1:(i-1)}$. (c) Generating relation-aware task-specific mask m^{T_i}.

respectively, which is defined as follows:

$$\mathbf{m}_k^{T_i} = \sigma(s \cdot \mathbf{e}_k^{T_i}) \tag{6}$$

where σ is the sigmoid function, and s is a positive scaling hyperparameter, which determines how much to amplify (or suppress) the layer output of significant (or insignificant) during training. Then, given the behavior sequence of user u_l, i.e., $\mathbf{x}^{u_l}$, we use the task-specific mask $\mathbf{m}_k^{T_i}$ to obtain the task-specific output of u_l in task T_i as follows:

$$\mathbf{E}_k^{u_l} = F_k(\mathbf{E}_{k-1}^{u_l}; \mathbf{m}_k^{T_i}) + \mathbf{E}_{k-1}^{u_l} = R_k(\mathbf{E}_{k-1}^{u_l}; \mathbf{m}_k^{T_i}) \tag{7}$$

where $F_k(\mathbf{E}_{k-1}^{u_l}; \mathbf{m}_k^{T_i})$ is the masked version of the residual mapping in layer k, obtained by element-wise multiplying the mask $\mathbf{m}_k^{T_i}$ (for convenience, we omit the notation of the layer indices in the residual block of the mask.) with each output of the ReLU activation in $F_k(\cdot)$, while R_k represents the residual block in layer k. We use $\mathcal{M}$ to denote the complete backbone network as well as the task embeddings from layer 1 to K, which is the single model we aim to learn to serve all tasks. The final output embedding of user u_l in task T_i, i.e., $\mathbf{E}_K^{u_l}$, is formulated as follows:

$$\mathbf{E}_K^{u_l} = \mathcal{M}(\mathbf{x}^{u_l}; \mathbf{m}^{T_i}) \tag{8}$$

where $\mathbf{m}^{T_i}$ is the task-specific mask of task T_i with a slight abuse of notation[2]. Note that although the user ID is given for each user $u_l \in \mathcal{U}$, it is not used to obtain $\mathbf{E}_K^{u_l}$.

Our proposed masking strategy defined in Equation 6 is different from that of CONURE defined in Equation 5 in two ways: (1) The mask of CONURE aims to freeze a small portion of parameters after having trained on a task, which gradually reduces the number of remaining available learnable parameters as more tasks are continually added. On the other hand, the mask of our proposed method does not freeze any of the parameters, thereby retaining the learning capability until the end of the training sequence. Moreover, we define our mask to be a *soft* continuous mask to allow all the parameters to be used for learning every task, whereas the mask of CONURE is a *hard* binary mask. (2) The mask of CONURE is applied on the actual model parameters in each layer (e.g., weights of convolutional filters in a CNN layer), whereas our mask is applied on the output of each layer. The major difference is in the size of the masks, i.e., the mask of CONURE is greatly larger than our mask. The reduced size of the mask not only enhances the model efficiency, but also enables our proposed method to better capture the relationship between tasks by obtaining a more compact representation of each task, i.e., $\mathbf{m}_k^{T_i}$.

4.2 Overcoming Catastrophic Forgetting via Knowledge Retention

However, as TERACON is not based on the parameter isolation approach, simply applying the masking strategy as described in Section 4.1 is prone to catastrophic forgetting, because we allow the parameters to be shared across tasks instead of freezing parameters used in previous tasks. Hence, to prevent catastrophic forgetting, we propose to transfer the knowledge of the current model regarding previous tasks to help train the current model itself. The key idea is, while training task T_i, to utilize the current backbone network $\mathcal{M}$, and $i-1$ task-specific classifiers of previous tasks, i.e., $G^{T_{1:(i-1)}}$, to generate the pseudo-labels for user $u_l \in \mathcal{U}^{T_i} \subset \mathcal{U}$ as follows:

$$\tilde{\mathbf{y}}_{u_l}^{T_j} = \tilde{G}^{T_j}(\tilde{\mathcal{M}}(\mathbf{x}^{u_l}; \tilde{\mathbf{m}}^{T_j})) \quad \text{for} \quad j = 1, \ldots, i-1 \tag{9}$$

where $\tilde{\mathbf{y}}_{u_l}^{T_j} \in \mathbb{R}^{|\mathcal{Y}^{T_j}|}$ is the pseudo-label of user u_l in task T_j, and $\tilde{G}$, $\tilde{\mathcal{M}}$, and $\tilde{\mathbf{m}}$ indicate frozen task-specific classifier, backbone network, and task-specific mask, respectively, which are only used to generate the pseudo-labels. Note that since each user $u_l \in \mathcal{U}^{T_i} \subset \mathcal{U}$ is represented by his/her behavior sequence in every task as described in Sec. 3.1, we can obtain the pseudo-label for each user in different tasks, even if user u_l is not observed in previous tasks $T_{1:(i-1)}$. Given the pseudo-labels, we minimize the following loss to retain the knowledge of previous tasks by training the current backbone model $\mathcal{M}$ to predict the pseudo-labels of u_l obtained from previous tasks $T_{1:(i-1)}$:

$$\mathcal{L}_{\text{KR}} = \mathop{\mathbb{E}}_{u_l \in \mathcal{U}^{T_i}} \left[\mathop{\mathbb{E}}_{1 \le j < i} \left[L_{\text{MSE}}(G^{T_j}(\mathcal{M}(\mathbf{x}^{u_l}; \mathbf{m}^{T_i})), \tilde{\mathbf{y}}_{u_l}^{T_j}) \right] \right] \tag{10}$$

where L_{MSE} is the mean squared error loss. That is, we continually update the backbone (i.e., $\mathcal{M}$) as well as the previous task-specific classifiers (i.e., $G^{T_{1:(i-1)}}$) based on the pseudo-labels generated from the backbone and the task-specific classifiers, which are obtained after the training of task T_{i-1} is finished, i.e., before we start training on the current task T_i. By doing so, TERACON retains the knowledge of the current model regarding the previous tasks, which helps to alleviate catastrophic forgetting.

4.3 Relation-aware Task-specific Mask

Recall that the second limitation of CONURE is that it fails to capture the relationship between tasks, since parameters used to learn previous tasks are kept frozen while learning new task. In this section, we describe how the relationship between tasks is captured by using the task-specific mask $\mathbf{m}_k^{T_i}$ computed in Equation 6. Specifically, when training on a new task, we aggregate the information from the previous tasks along with the current new task. Let T_i denote the current task, and assume that previous tasks, i.e., $T_{1:i-1}$,

[2] $\mathbf{m}^{T_i}$ subsumes all K masks from $\mathbf{m}_1^{T_i}$ to $\mathbf{m}_K^{T_i}$

have been trained by the model. We define an aggregate set of task T_j as follows:

$$\tilde{\mathcal{T}_j} = \{T_r | T_r \in T_{1:i}, \text{where } (T_i = \text{ current task}) \text{ and } (j \neq r)\} \tag{11}$$

where $\tilde{\mathcal{T}_j}$ is an aggregate set of task T_j. For example, given that we have already trained on T_1 and T_2, and the current task to be trained is the new task T_3, the aggregate sets for each task is given as: $\tilde{\mathcal{T}_1} : \{T_2, T_3\}$, $\tilde{\mathcal{T}_2} : \{T_1, T_3\}$, $\tilde{\mathcal{T}_3} : \{T_1, T_2\}$. Given the aggregate set of each task T_i, we update the task-specific mask $\mathbf{m}_k^{T_i}$ computed in Equation 6 to obtain the *relation-aware task-specific mask* as follows:

$$\mathbf{m}_k^{T_i} = \sigma\left(s \cdot f_k^{T_i}\left[\tanh(s \cdot \mathbf{e}_k^{T_i}) \parallel (\parallel_{T \in \tilde{\mathcal{T}_i}} \mathbf{p}_k^T)\right]\right) \in \mathbb{R}^f \tag{12}$$

where $\|$ is the vertical concatenation operation, $f_k^{T_i}$ is a 1-layer MLP, i.e., $\mathbb{R}^{(2|\tilde{\mathcal{T}}|+1)\times f} \longrightarrow \mathbb{R}^f$, which maps the concatenated task embeddings to a new task embedding of T_i, and $\mathbf{p}_k^T$ is defined as follows:

$$\mathbf{p}_k^T = \left[\tanh(s \cdot \mathbf{e}_k^T) \parallel \tanh(-s \cdot \mathbf{e}_k^T)\right] \in \mathbb{R}^{2\times f} \tag{13}$$

Recall that the task embedding $\mathbf{e}^T$ contains information about how much to amplify (or suppress) the layer output in task T. To provide a more nuanced understanding of task T, instead of directly using $\tanh(s \cdot \mathbf{e}_k^T)$, we introduce additional $\tanh(-s \cdot \mathbf{e}_k^T)$, which provides information about the opposite relatedness with $\tanh(s \cdot \mathbf{e}_k^T)$. Hence, $f_k^{T_i}$ learns to distinguish between the positive and negative information of tasks in $\tilde{\mathcal{T}_i}$. Moreover, it learns to determine which dimension of $\left[\tanh(s \cdot \mathbf{e}_k^{T_i}) \parallel (\parallel_{T \in \tilde{\mathcal{T}_i}} \mathbf{p}_k^T)\right]$ should send amplification or suppression signals to $\mathbf{m}_k^{T_i}$. In this situation, $\tanh(-s \cdot \mathbf{e}_k^T)$ serves as a counterbalance to the $\tanh(s \cdot \mathbf{e}_k^T)$, allowing $f_k^{T_i}$ to amplify the information of specific tasks (i.e., T), but also suppressing the same task if necessary. Therefore, this methods serves to provide a more complete and subtle fine-tune of representation for the new task embeddings. In summary, we obtain a relation-aware task-specific mask for task T_i, i.e., $\mathbf{m}_k^{T_i}$, by encoding the information regarding $\tilde{\mathcal{T}_i}$.

4.4 Relation-aware User Sampling Strategy

However, generating pseudo-labels and optimizing the loss $\mathcal{L}_{\text{KR}}$ defined in Equation 10 for each previous task using all the training data given in the current task is memory-inefficient and time-consuming. To alleviate the complexity issues, instead of using all the users in the current task, we propose to sample users from the current task that will be involved in the knowledge retention process. The key assumption is that retaining knowledge of a previous task would be easier if the current task is more similar to the previous task. Hence, we propose a sampling strategy that samples less users from the current task when addressing a previous task that is more similar to the current task. More precisely, given the current task T_i and a previous task $T_j \in T_{1:(i-1)}$, we sample a subset of users $\mathcal{U}_s^{T_i}$ uniformly at random from $\mathcal{U}^{T_i}$ as follows:

$$\mathcal{U}_s^{T_i}(j) \leftarrow \mathsf{sample}(\mathcal{U}^{T_i}, \rho_{i,j}) \tag{14}$$

where $\rho_{i,j}$ is the sampling rate that determines the amount of samples. Specifically, $(\rho_{i,j} \times 100)\%$ of users are sampled from $\mathcal{U}^{T_i}$, and $\rho_{i,j}$ is defined as follows:

$$\rho_{i,j} = 1 - \frac{1}{K}\sum_{k=1}^{K} \sigma(c \times \cos(\mathbf{m}_k^{T_i}, \tilde{\mathbf{m}}_k^{T_j})) \tag{15}$$

where $\cos(\cdot,\cdot)$ is cosine similarity, c is a scaling hyper-paramter, and $\tilde{\mathbf{m}}_k^{T_j}$ is the frozen mask of task T_j that is generated from the model before we start training the current task T_i. The main idea is to assign a small $\rho_{i,j}$ to a previous task T_j, if the similarity between T_j and the current task T_i is high, which encourages the model to sample a small number of users from similar tasks, while a large number of users are sampled from dissimilar tasks. Given sampled subsets of users for the current task T_i, we optimize the following loss:

$$\mathcal{L}_{\text{KR+}} = \mathop{\mathbb{E}}_{1 \le j < i}\left[\frac{\rho_{i,j}}{\sum_{k=1}^{i-1}\rho_{i,k}} \sum_{u_l \in \mathcal{U}_s^{T_i}(j)} \mathcal{L}_{\text{MSE}}(G^{T_j}(\mathcal{M}(\mathbf{x}^{u_l}; \mathbf{m}^{T_i})), \tilde{\mathbf{y}}_{u_l}^{T_j})\right] \tag{16}$$

By doing so, our model retains the knowledge from similar tasks efficiently and dissimilar tasks with a small amount of samples. We empirically observe in our experiments that the sampling ratio $\rho_{i,j}$ is within the range between 0.007 and 0.08, which implies that TERACON is efficiently trained in terms of both time and memory.

4.5 Training and Inference

4.5.1 *Training.*

For a given task T_i, TERACON is trained by optimizing the following objective function:

$$\mathcal{L} = \mathcal{L}_{\text{class}} + \alpha\mathcal{L}_{\text{KR+}} \tag{17}$$

where α is a coefficient that controls the contribution of the knowledge retention module. The classification loss, i.e., $\mathcal{L}_{\text{class}}$, is given as follows:

$$\mathcal{L}_{\text{class}} = \mathop{\mathbb{E}}_{u_l \in \mathcal{U}^{T_i}}\left[L_{\text{CE}}(G^{T_i}(\mathcal{M}(\mathbf{x}^{u_l}; \mathbf{m}^{T_i})), \mathbf{y}_{u_l}^{T_i})\right], \tag{18}$$

where L_{CE} is the cross-entropy loss, and $\mathbf{y}_{u_l}^{T_i} \in \{0,1\}^{|\mathcal{Y}^{T_i}|}$ is the ground truth one-hot label vector of user u_l in task T_i. Note that the above optimization is done for all M tasks in $\mathcal{T}$ in a sequential manner, i.e., $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_M$.

A Training Trick: Annealing s. In Eq. 6, when the positive scaling hyper-parameter s goes to infinity, i.e., $s \rightarrow \infty$, the task-specific mask operates like a step function i.e., $\mathbf{m}^{T_i} \rightarrow \{0,1\}^f$. This makes the gradient to flow into only a small portion of the task embedding $\mathbf{e}^{T_i}$ during training, which prevents an efficient training. To ensure a proper gradient flow on all task embeddings in the course of training, we introduce an annealing strategy [32] on s:

$$s = \frac{1}{s_{\max}} + (s_{\max} - \frac{1}{s_{\max}})\frac{b-1}{B-1} \tag{19}$$

where b is the batch index, B is the total number of batches in an epoch, and $s_{\max}$ is a positive scaling hyper-parameter. We generate masks with s during training and $s_{\max}$ during inference.

4.5.2 *Inference.*

Having trained all the tasks in $\mathcal{T}$, we perform inference on all the tasks in $\mathcal{T}$. Specifically, given a task $T_i \in \mathcal{T}$, we obtain the task-specific mask $\mathbf{m}^{T_i}$ based on the task embeddings as described in Eq. 12 and a task-specific classifier G^{T_i}, while fixing $s = s_{\max}$ in Eq. 19. Then, for each task $T_i \in \mathcal{T}$ and user $u_l \in \mathcal{U}^{T_i}$ whose behavior sequence is given by $\mathbf{x}^{u_l}$, we make the model predictions based on the task-specific user representation generated by utilizing the backbone network $\mathcal{M}$ as follows:

$$\hat{\mathbf{y}}_{u_l}^{T_i} = G^{T_i}(\mathcal{M}(\mathbf{x}^{u_l}; \mathbf{m}^{T_i})). \tag{20}$$

where $\hat{\mathbf{y}}_{u_l}^{T_i} \in \mathbb{R}^{|\mathcal{Y}^{T_i}|}$ is the prediction of labels of u_l in task T_i.

Table 1: Data Statistics ($|\mathcal{U}^{T_i}|$: num. users in T_i, $|\mathcal{Y}^{T_i}|$: num. unique labels in T_i).

Dataset	Task 1 (T_1)		Task 2 (T_2)		Task 3 (T_3)		Task 4 (T_4)		Task 5 (T_5)		Task 6 (T_6)	
	$\lvert\mathcal{U}^{T_1}\rvert$	$\lvert\mathcal{Y}^{T_1}\rvert$	$\lvert\mathcal{U}^{T_2}\rvert$	$\lvert\mathcal{Y}^{T_2}\rvert$	$\lvert\mathcal{U}^{T_3}\rvert$	$\lvert\mathcal{Y}^{T_3}\rvert$	$\lvert\mathcal{U}^{T_4}\rvert$	$\lvert\mathcal{Y}^{T_4}\rvert$	$\lvert\mathcal{U}^{T_5}\rvert$	$\lvert\mathcal{Y}^{T_5}\rvert$	$\lvert\mathcal{U}^{T_6}\rvert$	$\lvert\mathcal{Y}^{T_6}\rvert$
TTL	Watching		Clicking		Thumb-up		Age		Gender		Life status	
	1.47M	0.64M	1.39M	17K	0.25M	7K	1.47M	8	1.46M	2	1M	6
ML	Clicking		4-star		5-star		-		-		-	
	0.74M	54K	0.67M	26K	0.35M	16K						
NAVER Shopping	Search Query		Search Query		Item Category		Item Category		Gender		Age	
	0.9M	0.58M	0.59M	0.51M	0.15M	4K	0.15M	10	0.82M	2	0.82M	9

5 EXPERIMENTS

Datasets and Tasks. For comprehensive evaluations, we use two public datasets, i.e., Tencent TL (TTL) dataset[3] [42, 45] and Movielens (ML) dataset[4], and a proprietary NAVER Shopping dataset. Since there exists no dataset for continual user representation learning over tasks, we create various tasks for each public dataset following a previous work [45]. The detailed statistics for each dataset are described in Table 1, and task descriptions are provided as follows:

- **TTL dataset**[3][42, 45] consists of three item recommendation tasks and three user profiling tasks. Specifically, T_1 contains userID and the users' recent 100 news & video watching interactions on the QQ Browser platform. Based on the users' interaction history in QQ Browser platform, T_2 and T_3 aim to predict clicking interactions and thumb-up interactions on the Kandian platform, respectively. Moreover, in T_4, T_5, and T_6, models are trained to predict user's age, gender, and life status, respectively.
- **ML dataset**[4] consists of three tasks, all of which are related to item recommendation. Specifically, T_1 contains user IDs and their recent 30 clicking interactions, excluding the items that are 4-starred and 5-starred by the user. Given the recent 30 clicking interactions, models are trained to predict 4-starred and 5-starred items by the user in T_2 and T_3.
- **NAVER Shopping dataset** consists of two search query prediction tasks in the portal, two purchased item category prediction tasks in the shopping platform, and two user profiling tasks, which are elaborately designed considering the interests of the real-world industry. Specifically, T_1 includes userID and the user's recent 60 search queries in the online portal platform. Given the recent 60 search query histories, models are trained to predict next five search queries in T_2. Moreover, in T_3 and T_4, models are learn to predict the minor and major categories of user-purchased items based on the search query histories, which can also be recognized as cross-domain recommendations. Finally, the models are trained to predict users' gender in T_5 and age in T_6. We have detailed statistics and descriptions in Table 7 in Appendix A. We argue that obtaining the universal user representation from search queries, which is highly general in nature, is crucial in the online portal platforms since the various services can benefit from the highly transferrable representation. To the best of our knowledge, this is the first time the search queries are used to learn universal user representations in continual learning.

Methods Compared. We mainly compare TERACON to the most recent state-of-the-art recent method for learning universal user representation via continual learning, i.e., CONURE [45], and various other baseline methods, whose details are given as follows:

- **SinMo** trains a single model for each task from scratch, and thus no transfer learning occurs between tasks (M models in total).
- **FineAll** is pre-trained on task T_1 and fine-tuned on each task independently. Note that all the parameters in the backbone network and classifier layers are fine-tuned (M models in total).
- **PeterRec** [42] is pre-trained on the task T_1 and fine-tuned for each task with task-specific model patches and classifier layers. Differently from FineAll, PeterRec needs to maintain only a small number of parameters included in task-specific patches and classifiers, and they are fine-tuned in the subsequent tasks.
- **MTL** optimizes the model parameter via multi-task learning. Since not all users in T_1 exist in the remaining tasks, we conduct MTL with two objectives, i.e., one for T_1 and the other for $T_i (i > 1)$.
- **Piggyback** [23] / **HAT** [32] are the continual learning methods proposed in the computer vision domain, which we adapt to the universal user representation. **Piggyback** learns a binary mask for each task after obtaining the model parameters by pre-training on task T_1, and **HAT** learns a soft mask for the output of each layer and allows the entire model parameters to vary during the entire sequence of training tasks.
- **CONURE** [45] is our main baseline, which learns universal user representations based on the parameter isolation approach.

Evaluation Protocol. To evaluate the methods, we randomly split each dataset in T_i into train/validation/test data of 80/5/15% following previous work [45]. We use Mean Reciprocal Rank ($MRR@5$) to measure the model performance on item/query recommendation and product category prediction tasks, and the classification accuracy, i.e., $Acc = \frac{\text{\#Correct predictions}}{\text{\#Total number of users in task}}$, for user profiling tasks. We save model parameters for the next task when the performance on validation data gives the best result. Note that the performance of CL-based methods, i.e., Piggyback, HAT, CONURE, and TERACON, on each task is evaluated after continually training from T_1 to T_M, i.e., until the end of training sequence. Furthermore, we measure the effectiveness of capturing the relationship between tasks in terms of forward transfer (FWT) and backward transfer (BWT). More specifically, we use $FWT^{T_i} = \frac{R^{(T_i,T_i)} - \bar{R}^{T_i}}{\bar{R}^{T_i}} \times 100$ and $BWT^{T_i} = \frac{R^{(T_M,T_i)} - R^{(T_i,T_i)}}{R^{(T_i,T_i)}} \times 100$, where $R^{(T_j,T_i)}$ is the test performance of the model evaluated on T_i after training on T_j, where $j > i$, and $\bar{R}^{T_i}$ is test performance of **SinMo** on T_i. Note that $FWT^{T_i} > 0$ if the performance on T_i is better when it is evaluated after continually training from T_1 to T_i compared with the case when it is evaluated after training a single model on T_i from scratch. $BWT^{T_i} > 0$ if the performance on T_i is better when it is evaluated after continually training from T_1 to T_M, i.e., until the end of training sequence, compared with the case when it is evaluated after continually training from T_1 to T_i. We provide implementation details in Appendix C.

5.1 Overall Performance

The experimental results of sequential learning on the various tasks in three datasets are given in Table 2. We have the following observations: **1)** Positive transfer between tasks exists when comparing SinMo to other baseline methods (See also Table 3 (a)). Specifically, TL-based approaches, i.e., FineAll and PeterRec, outperform SinMo

[3] https://drive.google.com/file/d/1imhHUsivh6oMEtEW-RwVc4OsDqn-xOaP/view?usp=sharing

[4] https://grouplens.org/datasets/movielens/25m/

Table 2: Overall model performance over various tasks on TTL, ML, and NAVER Shopping datasets.

	TTL						ML			NAVER Shopping					
	T_1	T_2	T_3	T_4	T_5	T_6	T_1	T_2	T_3	T_1	T_2	T_3	T_4	T_5	T_6
SinMo	0.0446	0.0104	0.0168	0.4475	0.8901	0.4376	0.0566	0.0186	0.0314	0.0349	0.0265	0.0292	0.1984	0.5742	0.2985
FineAll	0.0446	0.0144	0.0218	0.5232	0.8851	0.4596	0.0566	0.0224	0.0328	0.0349	0.0318	0.0332	0.2367	0.6204	0.3247
PeterRec	0.0446	0.0147	0.0224	0.5469	0.8841	0.4749	0.0566	0.0224	0.0308	0.0349	0.0317	0.0322	0.2370	0.6257	0.3258
MTL	-	0.0102	0.0142	0.4672	0.8012	0.3993	-	0.0144	0.0267	-	0.0143	0.0266	0.1372	0.4998	0.2322
Piggyback	0.0446	0.0157	0.0236	0.5931	0.8990	0.5100	0.0566	0.0214	0.0302	0.0349	0.0314	0.0322	0.2349	0.6188	0.3129
HAT	0.0424	0.0174	0.0279	0.5880	0.9002	0.5126	0.0543	0.0227	0.0372	0.0344	0.0356	0.0317	0.2411	0.6294	0.3296
CONURE	0.0457	0.0169	0.0276	0.5546	0.8967	0.5230	**0.0598**	0.0244	0.0384	**0.0361**	0.0322	0.0305	0.2403	**0.6391**	0.3340
TERACON	**0.0474**	**0.0189**	**0.0316**	**0.6066**	**0.9048**	**0.5386**	0.0577	**0.0270**	**0.0459**	**0.0361**	**0.0359**	**0.0337**	**0.2444**	0.6381	**0.3354**

Table 3: Model performance on TTL dataset with the original and the reversed task sequence.

(a) Original	T_1			T_2			T_3			T_4			T_5			T_6		
	MRR@5	BWT	FWT	MRR@5	BWT	FWT	MRR@5	BWT	FWT	ACC	BWT	FWT	ACC	BWT	FWT	ACC	BWT	FWT
HAT	0.0424	-11.30%	-	0.0174	-7.45%	80.77%	0.0279	-0.71%	67.25%	0.5880	-2.52%	34.79%	0.9002	-1.98%	3.17%	0.5126	-	17.14%
CONURE	0.0457	-	-	0.0169	-	62.50%	0.0276	-	64.29%	0.5546	-	23.93%	0.8967	-	0.74%	0.5230	-	19.52%
TERACON	**0.0474**	-0.83%	-	**0.0189**	0.0%	81.73%	**0.0316**	3.27%	82.13%	**0.6066**	1.23%	33.91%	**0.9048**	0.01%	1.64%	**0.5386**	-	23.08%

(b) Reversed	T_1			T_6			T_5			T_4			T_3			T_2		
	MRR@5	BWT	FWT	ACC	BWT	FWT	ACC	BWT	FWT	ACC	BWT	FWT	MRR@5	BWT	FWT	MRR@5	BWT	FWT
HAT	0.0422	-11.72%	-	0.5025	-4.70%	20.49%	0.8980	-0.33%	1.22%	0.5770	-1.72%	31.19%	0.0269	-0.37%	60.71%	0.0184	-	76.92%
CONURE	0.0457	-	-	0.5322	-	21.62%	0.8849	-	-0.58%	0.5546	-	23.93%	0.0164	-	-2.38%	0.0119	-	14.42%
TERACON	**0.0474**	-0.83%	-	**0.5365**	1.84%	20.38%	**0.9039**	0.93%	0.61%	**0.6042**	0.07%	34.92%	**0.0313**	2.62%	81.55%	**0.0190**	-	82.69%

Table 4: Model performance and the performance degradation ratio (in bracket) compared to Table 2 (%) after training on a noisy task T'.

	TTL						
	T_1	T_2	T_3	T'	T_4	T_5	T_6
HAT	0.0411 (-3.06 %)	0.0165 (-5.17 %)	0.0259 (-7.16 %)	-	0.5424 (-7.76 %)	0.8870 (-1.47 %)	0.4873 (-4.94 %)
CONURE	0.0457 (0.0 %)	0.0169 (0.0 %)	0.0276 (0.0 %)	-	0.5245 (-5.43 %)	0.8663 (-3.39 %)	0.4469 (-14.55 %)
TERACON	**0.0472** (-0.42 %)	**0.0189** (0.0 %)	**0.0314** (-0.63 %)	-	**0.6022** (-0.73 %)	**0.9014** (-0.38 %)	**0.5312** (-1.37 %)

	NAVER Shopping						
	T_1	T_2	T'	T_3	T_4	T_5	T_6
HAT	0.0314 (-8.72%)	0.0302 (-15.16%)	-	0.0309 (-2.52%)	0.2357 (-2.24%)	0.6219 (-1.19%)	0.3180 (-3.51%)
CONURE	**0.0361** (0.0%)	0.0322 (0.0%)	-	0.0291 (-4.59%)	0.2231 (-7.16%)	0.6202 (-2.95%)	0.3122 (-6.53%)
TERACON	0.0346 (-4.15%)	**0.0336** (-6.41%)	-	**0.0329** (-2.37%)	**0.2378** (-2.7%)	**0.6348** (-0.52%)	**0.3329** (-0.75%)

in most of the cases, indicating that the knowledge obtained from task T_1 is helpful in learning task T_i $(i > 1)$. Moreover, CL-based methods, i.e., HAT, CONURE, and TERACON, outperform the TL-based methods verifying that the knowledge transfer happens not only between paired tasks but also between multiple tasks. On the other hand, positive transfer rarely happens when multiple tasks are simultaneously trained (See SinMo vs. MTL). These results indicate that the online web platform, which contains multiple tasks in nature, requires continual user representations for the true understanding of users. **2)** TERACON outperforms the CL-based approaches that do not consider the relationship between the tasks, i.e., Piggyback, HAT, and CONURE. This is because learning the relationship between the tasks, i.e., whether the tasks are positively related or not, can further accelerate the knowledge transfer between the tasks, which has been shown to exist in our first observation above. **3)** Moreover, we observe a positive backward transfer occurs when training TERACON in Table 3 (a), which can never happen in parameter isolation-based approaches, i.e., CONURE (hence, BWT of CONURE is not reported). This is because our model allows the entire model parameters to be modified during the entire training sequence, enabling the knowledge obtained from the new tasks to be transferred to the previous tasks. **4)** However, allowing the model parameters to be modified also incurs a severe performance degradation as a new task arrives, i.e., catastrophic forgetting, if there exists no module specifically designed to alleviate the issue (see HAT in Table 3 (a)). This also verifies that TERACON successfully alleviates catastrophic forgetting with the knowledge retention module, which facilitates the model to retain the knowledge obtained from the previous tasks.

5.2 Comparison to Parameter Isolation

In this section, we verify the effectiveness of TERACON compared to the parameter isolation-based approach, i.e., CONURE, by conducting experiments on the different sequences of tasks in Table 3, and inserting noisy tasks among the sequence of tasks in Table 4.

TERACON is robust to the change of task orders. As shown in Table 3 (b), our model maintains its performance even when the task sequence is trained in the reversed direction, i.e., $T_1 \rightarrow T_6 \rightarrow T_5 \rightarrow \cdots \rightarrow T_2$, while the CONURE's performance in task T_2 and T_3 severely deteriorates. As we mentioned in Sec. 3.4, this is because the number of trainable parameters in CONURE decreases as the number of trained tasks increases. Therefore, when a large portion of model parameters are trained for learning less informative tasks in the early stage, the model cannot learn the subsequent tasks that actually require a large number of parameters for generalization. On the other hand, TERACON successfully learns the subsequent tasks by maintaining its learning capacity during the entire sequence of training. Considering that the sequence of tasks cannot be arbitrarily determined in the real world, we argue that TERACON is more practical than CONURE in real-world applications.

TERACON is robust to the negative transfer. To investigate the impact of negative transfer between the tasks, we conduct experiments by inserting an uninformative task among the sequence of tasks. Specifically, given the total user set $\mathcal{U}$, we randomly sample 50% of users and generate a random label of 50 classes for each user to create a noisy task T'. In Table 4, we have the following observations: **1)** Compared to Table 2, the overall performance of all methods degrade due to the effect of noisy task T'. This indicates that a negative transfer occurs between tasks that have no relationship between one another. However, thanks to the parameter isolation approach, CONURE does not experience a performance drop in the tasks trained before the noisy task. **2)** On the other hand, we observe that CONURE suffers from the largest performance drop among the methods in the tasks that are trained after the noisy task. This is because the model parameters that are disrupted while training the noisy task are frozen after the task, impacting the performance of the subsequent tasks. **3)** Moreover, we observe that TERACON successfully alleviates the effect of the noisy task with a moderate performance drop, which is in contrast to HAT. We attribute this to our proposed relation-aware task-specific masks, which automatically disregard the information from the noisy task making TERACON robust to the negative transfer.

To summarize our findings, TERACON is robust to the change of task orders and the negative transfer from noisy tasks, demonstrating the importance of 1) maintaining the learning capacity and 2) the relation-aware task-specific masks. Moreover, considering that there exist multiple tasks of unknown sequence or negative correlation in real-world web platforms, we argue that TERACON is practical in reality. We further analyze the universal user representation obtained by the methods Figure 2 in Appendix B.2.

5.3 Model Analysis

Effect of Relation-aware Task-specific Masks. We investigate the importance of the relation-aware task-specific masks by comparing various masking strategies. Specifically, we compare the proposed relation-aware task-specific masks defined in Eq. 12 with the masks that do not aggregate the information from the other tasks, i.e., *w/o relation*, defined in Eq. 6, and the mask without the opposite relatedness, i.e., *only positive* (no $\tanh(-s \cdot \mathbf{e}_k^T)$ in Eq. 13). Based on Table 5, we have the following observations: **1)** The masks that do not have the task-relation information between the tasks perform the worst (i.e., *w/o relation*), indicating that injecting the relationship between tasks into the task-specific masks is helpful for transferring knowledge between the tasks. **2)** On the other hand, by comparing the BWT ratio between *only positive* and TERACON, we observe that modeling the opposite relatedness between the tasks (see Eq. 13) is crucial for alleviating catastrophic forgetting. This is because the opposite relatedness provides further relational information between the tasks, making it easy for the model to decide which dimension of the task embedding to amplify or suppress. In summary, our proposed relation-aware task-specific masks not only successfully transfer the knowledge between the tasks but also retain the previously obtained knowledge. Refer to Figure 3 in Appendix B.3 for other masking strategies.

Table 5: Model performance and BWT ratio (in bracket) with various masking strategies on TTL dataset.

	T_1	T_2	T_3	T_4	T_5	T_6
w/o relation	0.0455 (-4.81%)	0.0182 (-1.09%)	0.0292 (-1.02%)	0.5940 (-0.26%)	0.8999 (-0.06%)	0.5354 (0.0%)
only positive	0.0471 (-1.46%)	0.0185 (-1.07%)	0.0293 (0.69%)	0.6054 (-1.11%)	0.9023 (0.21%)	0.5289 (0.0%)
TERACON	**0.0474** (-0.83%)	**0.0189** (0.0%)	**0.0316** (3.27%)	**0.6066** (1.23%)	**0.9048** (0.01%)	**0.5386** (0.0%)

Table 6: Model performance and training time (i.e., sec/epoch) (in bracket) over various sampling strategies on TTL dataset.

	Sampling	T_1	T_2	T_3	T_4	T_5	T_6
$\rho_{i,j} = \rho_{min}$	✓	0.0470 (-)	0.0184 (625.47)	0.0280 (77.82)	0.6027 (417.65)	0.9007 (510.80)	0.5385 (414.44)
$\rho_{i,j} = $ Eq.15	✓	0.0474 (-)	0.0189 (625.47)	**0.0316** (90.79)	0.6066 (504.3)	**0.9048** (583.77)	0.5386 (494.14)
$\rho_{i,j} = 1.0$	✗	**0.0475** (-)	**0.0190** (1146.70)	0.0313 (151.32)	**0.6143** (1179.31)	0.9047 (1355.18)	**0.5403** (797.09)

Effect of Relation-aware User Sampling. We verify the effectiveness of the user sampling in terms of the model performance and training time in Table 6. To do so, we compare the performance of the automatic sampling strategy described in Eq. 14 with a variant user sampling strategy, i.e., $\rho_{i,j} = \rho_{min}$ that selects the minimum value among $\rho_{i,j}$ for all i and j when training task T_i. Moreover, we compare it to the one that generates pseudo-labels for all the users in the task, i.e., no sampling, which is equivalent to setting $\rho_{i,j} = 1.0$. In Table 6, we observe that $\rho_{i,j} = 1.0$ generally performs the best, which is expected as all the users in every task is used to train the model. However, we observe that our proposed sampling strategy in Eq. 14 shows a competitive performance even with a small number of users in each task and reduced training time, which shows the efficiency of our proposed relation-aware user sampling strategy[5]. Finally, $\rho_{i,j} = \rho_{min}$ performs the worst, which implies that considering the relationship between the tasks is beneficial when sampling users. Refer to Table 8, 9 in Appendix B.1 for additional experiments on user sampling.

6 CONCLUSION

In this paper, we propose a novel continual user representation learning model, named TERACON. The main idea is to utilize task embeddings for generating relation-aware task-specific masks that enable the model to maintain the learning capability during training and facilitate the relationship between the tasks to be captured. By doing so, TERACON successfully transfers the knowledge between the tasks not only in the forward direction but also backward direction. Moreover, TERACON prevents catastrophic forgetting through a novel knowledge retention module. Our extensive experiments on various real-world datasets demonstrate that TERACON is not only effective and efficient, but also robust to the change of task orders and negative transfer, which shows the practicality of TERACON in real-world applications.

Acknowledgements This work was supported by NAVER Corporation, and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2022-0-00157).

[5] As mentioned in Sec. 4.4, the value of $\rho_{i,j}$ is between 0.0007 and 0.08, which means only 0.7% to 8% of the users are required to achieve the performance shown in Table 6.

REFERENCES

[1] Magdalena Biesialska, Katarzyna Biesialska, and Marta R Costa-Jussa. 2020. Continual lifelong learning in natural language processing: A survey. *arXiv preprint arXiv:2012.09823* (2020).

[2] Michael Chui, James Manyika, Mehdi Miremadi, Nicolaus Henke, Rita Chung, Pieter Nel, and Sankalp Malhotra. 2018. Notes from the AI frontier: Insights from hundreds of use cases. *McKinsey Global Institute* (2018), 28.

[3] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems* (Boston, Massachusetts, USA) *(RecSys '16)*. Association for Computing Machinery, New York, NY, USA, 191–198. https://doi.org/10.1145/2959100.2959190

[4] Michael Crawshaw. 2020. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796* (2020).

[5] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. 2021. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence* 44, 7 (2021), 3366–3385.

[6] Jie Gu, Feng Wang, Qinghui Sun, Zhiquan Ye, Xiaoxiao Xu, Jingmin Chen, and Jun Zhang. 2021. Exploiting behavioral consistence for universal user representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 4063–4071.

[7] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. *arXiv preprint arXiv:1703.04247* (2017).

[8] Xiangnan He and Tat-Seng Chua. 2017. Neural Factorization Machines for Sparse Predictive Analytics *(SIGIR '17)*. Association for Computing Machinery, New York, NY, USA, 355–364. https://doi.org/10.1145/3077136.3080777

[9] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[10] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 2790–2799. https://proceedings.mlr.press/v97/houlsby19a.html

[11] Ching-Yi Hung, Cheng-Hao Tu, Cheng-En Wu, Chien-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. 2019. Compacting, picking and growing for unforgetting continual learning. *Advances in Neural Information Processing Systems* 32 (2019).

[12] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*. IEEE, 197–206.

[13] Zixuan Ke, Bing Liu, and Xingchang Huang. 2020. Continual learning of a mixed sequence of similar and dissimilar tasks. *Advances in Neural Information Processing Systems* 33 (2020), 18493–18504.

[14] Kibum Kim, Dongmin Hyun, Sukwon Yun, and Chanyoung Park. 2023. MELT: Mutual Enhancement of Long-Tailed User and Item for Sequential Recommendation. *arXiv preprint arXiv:2304.08382* (2023).

[15] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* 114, 13 (2017), 3521–3526.

[16] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37. https://doi.org/10.1109/MC.2009.263

[17] Lilly Kumari, Shengjie Wang, Tianyi Zhou, and Jeff Bilmes. 2022. Retrospective Adversarial Replay for Continual Learning. In *Advances in Neural Information Processing Systems*.

[18] Sebastian Lee, Sebastian Goldt, and Andrew Saxe. 2021. Continual learning in the teacher-student setup: Impact of task similarity. In *International Conference on Machine Learning*. PMLR, 6109–6119.

[19] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. 2020. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information fusion* 58 (2020), 52–68.

[20] Sheng Li, Jaya Kawale, and Yun Fu. 2015. Deep Collaborative Filtering via Marginalized Denoising Auto-Encoder *(CIKM '15)*. Association for Computing Machinery, New York, NY, USA, 811–820. https://doi.org/10.1145/2806416.2806527

[21] Sheng Li and Handong Zhao. 2021. A Survey on Representation Learning for User Modeling. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence* (Yokohama, Yokohama, Japan) *(IJCAI'20)*. Article 695, 7 pages.

[22] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H. Chi. 2018. Modeling Task Relationships in Multi-Task Learning with Multi-Gate Mixture-of-Experts. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 1930–1939. https://doi.org/10.1145/3219819.3220007

[23] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. 2018. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 67–82.

[24] Arun Mallya and Svetlana Lazebnik. 2018. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 7765–7773.

[25] Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Vol. 24. Elsevier, 109–165.

[26] Yabo Ni, Dan Ou, Shichen Liu, Xiang Li, Wenwu Ou, Anxiang Zeng, and Luo Si. 2018. Perceive Your Users in Depth: Learning Universal User Representations from Multiple E-commerce Tasks. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018).

[27] Yabo Ni, Dan Ou, Shichen Liu, Xiang Li, Wenwu Ou, Anxiang Zeng, and Luo Si. 2018. Perceive your users in depth: Learning universal user representations from multiple e-commerce tasks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 596–605.

[28] Roger Ratcliff. 1990. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review* 97, 2 (1990), 285.

[29] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting Clicks: Estimating the Click-through Rate for New Ads. In *Proceedings of the 16th International Conference on World Wide Web* (Banff, Alberta, Canada) *(WWW '07)*. Association for Computing Machinery, New York, NY, USA, 521–530. https://doi.org/10.1145/1242572.1242643

[30] Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* (2017).

[31] Ruslan Salakhutdinov and Andriy Mnih. 2007. Probabilistic Matrix Factorization. In *Proceedings of the 20th International Conference on Neural Information Processing Systems* (Vancouver, British Columbia, Canada) *(NIPS'07)*. Curran Associates Inc., Red Hook, NY, USA, 1257–1264.

[32] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. 2018. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*. PMLR, 4548–4557.

[33] Xiang-Rong Sheng, Liqin Zhao, Guorui Zhou, Xinyao Ding, Binding Dai, Qiang Luo, Siran Yang, Jingshan Lv, Chi Zhang, Hongbo Deng, et al. 2021. One model to serve all: Star topology adaptive recommender for multi-domain ctr prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 4104–4113.

[34] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. 2017. Continual learning with deep generative replay. *Advances in neural information processing systems* 30 (2017).

[35] Qinghui Sun, Jie Gu, Bei Yang, XiaoXiao Xu, Renjun Xu, Shangde Gao, Hong Liu, and Huan Xu. 2021. Interest-oriented universal user representation via contrastive learning. *arXiv preprint arXiv:2109.08865* (2021).

[36] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[37] Hong Wen, Jing Zhang, Yuan Wang, Fuyu Lv, Wentian Bao, Quan Lin, and Keping Yang. 2020. Entire Space Multi-Task Modeling via Post-Click Behavior Decomposition for Conversion Rate Prediction *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 2377–2386. https://doi.org/10.1145/3397271.3401443

[38] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 346–353.

[39] Xiangli Yang, Qing Liu, Rong Su, Ruiming Tang, Zhirong Liu, and Xiuqiang He. 2021. Autoft: Automatic fine-tune for parameters transfer learning in click-through rate prediction. *arXiv preprint arXiv:2106.04873* (2021).

[40] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. 2021. Online coreset selection for rehearsal-based continual learning. *arXiv preprint arXiv:2106.01085* (2021).

[41] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. 2017. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547* (2017).

[42] Fajie Yuan, Xiangnan He, Alexandros Karatzoglou, and Liguang Zhang. 2020. Parameter-efficient transfer from sequential behaviors for user modeling and recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 1469–1478.

[43] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. 2019. A simple convolutional generative network for next item recommendation. In *Proceedings of the twelfth ACM international conference on web search and data mining*. 582–590.

[44] Feng Yuan, Lina Yao, and Boualem Benatallah. 2019. DARec: Deep domain adaptation for cross-domain recommendation via transferring rating patterns. *arXiv preprint arXiv:1905.10760* (2019).

[45] Fajie Yuan, Guoxiao Zhang, Alexandros Karatzoglou, Joemon Jose, Beibei Kong, and Yudong Li. 2021. One person, one model, one world: Learning continual user representation without forgetting. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 696–705.

[46] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. 2018. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3712–3722.

[47] Lili Zhao, Sinno Jialin Pan, Evan Wei Xiang, Erheng Zhong, Zhongqi Lu, and Qiang Yang. 2013. Active Transfer Learning for Cross-System Recommendation. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence* (Bellevue, Washington) *(AAAI'13)*. AAAI Press, 1205–1211.

[48] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 5941–5948.

[49] Yin Zhu, Yuqiang Chen, Zhongqi Lu, Sinno Jialin Pan, Gui-Rong Xue, Yong Yu, and Qiang Yang. 2011. Heterogeneous Transfer Learning for Image Classification. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence* (San Francisco, California) *(AAAI'11)*. AAAI Press, 1304–1309.

A NAVER SHOPPING DATASET

In this paper, we use a proprietary dataset that is collected at an e-commerce platform, i.e., NAVER Shopping, from 07/01/2022 to 11/30/2022. We randomly selected a subset of users from that period whose search query history is longer than 10, and the queries that are searched more than 200 times. That is, user search behavior sequences used in this paper have queries searched over 200 times and a user sequence length greater than 10. The e-commerce platform contains a multitude of services that are not limited to online shopping, e.g., search engines and news. We process the datasets to create a sequence of tasks, regarding the interest and needs of the platform. NAVER Shopping consists of two search query prediction tasks in the portal whose queries are not limited to the shopping domain, two purchased item category prediction tasks in the shopping platform, and two tasks for predicting users' gender and age. We have detailed statistics and descriptions in Table 7 and below:

Table 7: Statistics for NAVER Shopping dataset.

	Task	# Users	# Unique labels	Date
T_1	Next Search Query	0.9M	0.58M	07/01/2022 ~ 10/31/2022
T_2	Next Search Query	0.59M	0.51M	11/01/2022 ~ 11/30/2022
T_3	Minor Item Category	0.15M	4K	11/01/2022 ~ 11/30/2022
T_4	Major Item Category	0.15M	10	11/01/2022 ~ 11/30/2022
T_5	Gender	0.82M	2	-
T_6	Age	0.82M	9	-

- T_1 consists of the users' recent search queries in the online portal platform during the period of 07/01/2022 to 10/31/2022. We use a user's 60 recent search queries, and train the model to predict the next search query in an autoregressive manner.
- T_2 consists of the next five search queries of the search queries contained in T_1. That is, search queries in T_2 are collected during the period of 11/01/2022 to 11/30/2022. Given the recent 60 search queries of the user in T_1, the model is trained to predict the user's next five search queries.
- T_3 and T_4 are item category prediction tasks with a different hierarchy. Note that the items in NAVER Shopping dataset are hierarchically categorized, i.e., **major** categories → **middle** categories → **small** categories → **minor** categories. Therefore, given a user's 60 recent search queries, the model is trained to predict **minor** and **major** categories of the items during the period of 2022/11/01 to 2022/11/30 in T_3 and T_4, respectively. It is also worth noting that T_3 and T_4 can be recognized as cross-domain recommendations, which is a common interest in industry, since the model aims to recommend **shopping items** based on the user's **search query history**.
- T_5 is a gender prediction task based on the user's recent 60 search queries.
- T_6 is an age prediction task based on the user's recent 60 search queries.

B ADDITIONAL EXPERIMENTS

B.1 Effect of Relation-aware User Sampling

In Table 8, we additionally conduct experiments on the effect of relation-aware user sampling in NAVER Shopping dataset. We observe that TERACON efficiently learns from a portion of the users with a moderate performance degradation.

Table 8: Model performance and training time (i.e., sec/epoch) (in bracket) over various sampling ratios on NAVER Shopping dataset.

	T_1	T_2	T_3	T_4	T_5	T_6
ρ_{min}	0.0358 (-)	0.0350 (417.71)	0.0323 (111.60)	0.2433 (62.06)	0.6379 (215.25)	0.3352 (248.26)
w/o sampling	**0.0363** (-)	0.0361 (626.57)	0.0342 (172.41)	0.2485 (133.45)	0.6401 (428.74)	0.3352 (446.47)
TERACON	0.0361 (-)	0.0359 (417.71)	0.0337 (135.13)	0.2444 (79.56)	0.6381 (242.16)	0.3354 (278.95)

Moreover, we also investigate the effect of relation-aware user sampling in the TTL dataset of reverse order in Table 9. We observe that relation-aware user sampling approaches also maintain the model's robustness to the task orders.

Table 9: Model performance and training time (i.e., sec/epoch) (in bracket) over various sampling ratios on TTL dataset Reversed task sequence.

	Sampling	T_1	T_6	T_5	T_4	T_3	T_2
$\rho_{i,j} = \rho_{min}$	✓	0.0471 (-)	0.5304 (393.53)	0.9010 (491.11)	0.6011 (449.84)	0.0307 (92.44)	0.0188 (743.94)
$\rho_{i,j}$ = Eq.15	✓	0.0474 (-)	0.5365 (393.53)	**0.9039** (548.92)	0.6042 (517.22)	**0.0313** (108.98)	0.0190 (873.98)
$\rho_{i,j}$ = 1.0	✗	**0.0475** (-)	**0.5366** (568.11)	0.9031 (989.42)	**0.6104** (1133.54)	0.0311 (197.68)	**0.0192** (1673.55)

B.2 Evaluating Universal User Representation

In this section, we empirically evaluate the quality of universal user representations by comparing how the representation adapts to the new tasks in Figure 2. More specifically, we train the models on the sequence of tasks T_1, T_2, and T_3 in TTL dataset, and then evaluate how the embeddings obtained from each model adapt to the task T_4, i.e., age prediction task. We have the following observations: **1)** By comparing the initial performance between the methods, we argue that the universal user representation obtained by TERACON provides better initial points for the new tasks compared to the baseline methods. **2)** Moreover, thanks to the good initial points, our model converges to the optimal point much faster than the baseline methods. Therefore, we argue that TERACON truly learns the universal user representation that can be easily adapted to the various tasks, which will bring further practicality to web platform applications.

B.3 Advantages of Masking Layer Output

In this section, we investigate the advantages of masking the outputs of each layer compared to directly masking the model parameters by measuring the test performance during the whole training epochs in Figure 3. We observe that model that masks the whole

Table 10: Hyperparameter specifications of TERACON

	TTL						ML			NAVER Shopping					
	T_1	T_2	T_3	T_4	T_5	T_6	T_1	T_2	T_3	T_1	T_2	T_3	T_4	T_5	T_6
Learning rate (η)	0.01	0.0001	0.0001	0.0001	0.0001	0.0001	0.001	0.0001	0.0005	0.001	0.0001	0.0001	0.0005	0.0005	0.0005
Scaling (c)	-	6.0	6.0	6.0	6.0	6.0	-	6.0	6.0	-	6.0	6.0	6.0	6.0	6.0
Positive scaling ($s_{\max}$)	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
Knowledge retention (α)	-	0.7	0.7	0.7	0.7	0.7	-	0.7	0.9	-	0.9	0.9	0.7	0.7	0.9
Batch size (b)	32	1024	1024	1024	1024	1024	32	1024	1024	64	1024	1024	1024	1024	1024

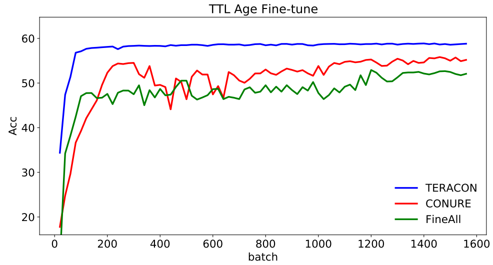

Figure 2: Model performance per epoch on the age prediction task.

parameters, i.e., Parameter, has a worse initial point and converges slowly compared to TERACON. This is because masking all parameters require the same number of parameters for masking since the masking operation is done via an elementwise product operation, which will make it difficult to train the model. On the other hand, TERACON requires a small portion of parameters for masking, which is beneficial in model training and computational cost.

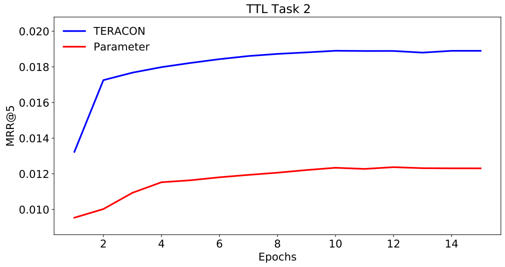

Figure 3: Comparison on the parameter masking and TERACON.

C IMPLEMENTATION DETAILS

For a fair comparison, we set the dimension of item and task embeddings to 256 for all the methods and datasets. All the tasks are conducted 1024 batch size (b) but for T_1, due to the limitations of GPU memory, we use a smaller batch size (b). Moreover, we use the Adam optimizer to train the models in all tasks. For hyperparameters, we tune the model in certain ranges as follows: learning rate η in $\{0.001, 0.0005, 0.0001, 0.00005\}$, scaling c in $\{3.0, 4.5, 6.0\}$, positive scaling $s_{\max}$ in $\{5, 50, 100, 500\}$, knowledge retention coefficient α in $\{0.1, 0.3, 0.5, 0.7, 0.9, 1.0\}$. We report the best performing hyperparameters for all the tasks in each dataset in Table 10.

For the baseline methods, we tried our best to reproduce the results in their own papers by following their desciption on implementation details. Specifically, we follow the pruning ratios reported on CONURE for each task, i.e., 70/80/90/80/90/90% and 70/80/70%, for TTL and ML, respectively. Moreover, for NAVER Shopping dataset, we tune the pruning ratio between 70 ∼ 90% for each task.

We use NVIDIA GeForce A6000 48GB for TTL and ML dataset, and use eight Tesla P40 for NAVER Shopping datasets.

C.1 Reproducibility

For baseline models, we use the official codes published by authors as shown in Table 11, and then conduct evaluations within the same environment. Refer to ***Souce code link*** for our source code and instructions on how to run the code to reproduce the results reported in the experiments.

Table 11: Source code links of the baseline methods.

Methods	Source code
CONURE	https://github.com/yuangh-x/2022-NIPS-Tenrec
PeterRec	https://github.com/yuangh-x/2022-NIPS-Tenrec
HAT	https://github.com/joansj/hat
Piggyback	https://github.com/arunmallya/piggyback
NextitNet	https://github.com/syiswell/NextItNet-Pytorch
TERACON	https://github.com/Sein-Kim/TERACON