One Person, One Model, One World: Learning Continual User Representation without Forgetting

Fajie Yuan[†,+], Guoxiao Zhang[+], Alexandros Karatzoglou[‡], Joemon Jose[≀], Beibei Kong[+], Yudong Li[+]
[†]Westlake University, China [+]Tencent, China, [‡]Google, UK [≀]University of Glasgow, UK
yuanfajie@westlake.edu.cn,suranzhang@tencent.com,alexandros.karatzoglou@gmail.com

ABSTRACT

Learning user representations is a vital technique toward effective user modeling and personalized recommender systems. Existing approaches often derive an individual set of model parameters for each task by training on separate data. However, the representation of the same user potentially has some commonalities, such as preference and personality, even in different tasks. As such, these separately trained representations could be suboptimal in performance as well as inefficient in terms of parameter sharing.

In this paper, we delve on research to continually learn user representations task by task, whereby new tasks are learned while using partial parameters from old ones. A new problem arises since when new tasks are trained, previously learned parameters are very likely to be modified, and as a result, an artificial neural network (ANN)-based model may lose its capacity to serve for well-trained previous tasks forever, this issue is termed catastrophic forgetting. To address this issue, we present *Conure* the first continual, or lifelong, user representation learner — i.e., learning new tasks over time without forgetting old ones. Specifically, we propose iteratively removing less important weights of old tasks in a deep user representation model, motivated by the fact that neural network models are usually over-parameterized. In this way, we could learn many tasks with a single model by reusing the important weights, and modifying the less important weights to adapt to new tasks. We conduct extensive experiments on two real-world datasets with nine tasks and show that *Conure* largely exceeds the standard model that does not purposely preserve such old "knowledge", and performs competitively or sometimes better than models which are trained either individually for each task or simultaneously by merging all task data.

CCS CONCEPTS

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

KEYWORDS

User Modeling; Lifelong Learning; Forgetting; Recommender Systems

ACM Reference Format:
Fajie Yuan[†,+], Guoxiao Zhang[+], Alexandros Karatzoglou[‡], Joemon Jose[≀], Beibei Kong[+], 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 (SIGIR '21), July 11–15, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3404835.3462884

1 INTRODUCTION

In the last decade, social medial and e-commerce systems, such as TikTok, Facebook and Amazon, have become increasingly popular and gained success due to the convenience they provide in people's lives. For example, as the biggest social network, Facebook has over 2.6 billion monthly active users.[1] On the other hand, a large number of user behavior feedback (e.g., clicks, likes, comments and shares) is created every day on these systems. An impressive example is TikTok, where users can easily watch hundreds of short videos per day given that the play duration per video takes usually less than 30 seconds [42].

A large body of research [4, 9, 12, 32, 38, 40, 43, 46] has demonstrated that the user behavior signals can be used to model their preference so as to provide personalized services, e.g., for recommender systems. However, most of these work focuses only on the tasks of user modeling (UM) or item recommendation on the same platform, from where the data comes. Unlike these works, recently [42] took an important step, which revealed that the user representations learned from an upstream recommendation task could be a generic representation of the user and could be directly transferred to improve a variety of dissimilar downstream tasks. To this end, they proposed a two-stage transfer learning paradigm, termed PeterRec, which first performs self-supervised pretraining on user behavior sequences, and then performs task-specific supervised finetuning on the corresponding downstream tasks.

Despite that PeterRec has achieved some positive transfer, the downstream tasks it served for, however, are trained individually. These tasks may share substantial similarities in practice if the same users are involved. E.g., users who retweet a message posted on Twitter tend to give it a thumb-up as well. That is, the task of thumb-up prediction has some correlations with the task of retweet prediction. Arguably, we believe learning user representations from many tasks is important and could potentially obtain better performance on related tasks. Besides, training tasks individually requires

[†,+]This work was done when Fajie worked at Tencent (past affiliation) and Westlake University (current affiliation).

[1] https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide

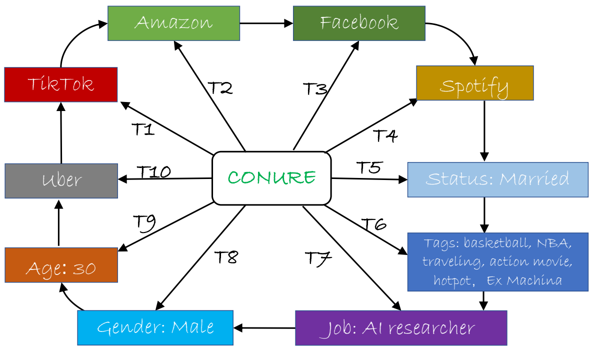


Figure 1: Illustration of one person, one model, one world. 'T' is short for task (e.g., item recommendation (i.e., T1, T2, T3, T4, T10) or profile prediction (i.e., T5, T6, T7, T8, T9)). While a person played different roles in the 10 scenarios in his (real or virtual) world, a lifelong UR model, for example, *Conure*, should be able to learn user representations continually task by task and then serve all of them after a round of training.

additional storage overhead to keep parameters per model per task; otherwise, training them one by one with a single model may lead to catastrophic forgetting [17, 20]. In parallel, another line of work usually perform multi-task learning (MTL) on related tasks [19, 26]. This could be beneficial as well but sometimes infeasible since training data for all tasks might not always be simultaneously available. Moreover, some incoming tasks may not have overlapping users, making such joint learning-based methods often infeasible.

To deal with the above-mentioned issues, we explore a promising but more challenging learning paradigm for user modeling — i.e., lifelong user representation (UR) learning over tasks. Our goal is to develop an artificial neural network (ANN)-based UR model that not only provides universal user representations, but also has the continuous learning ability throughout its lifespan: quickly learning new abilities based on previously acquired knowledge, and being immune to forgetting old knowledge. Moreover, the proposed UR model should build up and modify representations for each person with only one backbone network architecture, whereby all roles the person played in their individual (real or virtual) world can be well described. Ideally, with such a comprehensive UR model, our understanding towards user needs, preferences, cognitive and behavioural characteristics could enter a new stage. We refer to this goal as One Person, One Model, One World, as shown in Figure 1.

To motivate this work, we first perform ablation studies to show two unexplored phenomena for deep UR models: i) sequentially learning different tasks and updating parameters for a single ANN-based UR model leads to catastrophic forgetting, and correspondingly, the UR model loses its prediction ability for old tasks that were trained before; ii) removing a certain percentage of unimportant/redundant parameters for a well-trained deep UR model does not cause irreversible degradation on its prediction accuracy. Taking inspiration from the two insights, we propose a novel continual, or lifelong, user representation learning framework, dubbed as *Conure*. *Conure* is endowed the lifelong learning capacity for a number of tasks related to user profile prediction and item recommendation, where it addresses the forgetting issue for old tasks by important knowledge retention, and learns new tasks by exploiting parameter redundancy. We summarize our main contributions as follows.

- We open a new research topic and formulate the first UR learning paradigm that deals with a series of ***different*** tasks coming either sequentially or separately.[2] Besides, we show in-depth empirical analysis for the forgetting problem and network redundancy in deep UR models under the proposed lifelong learning setting.
- We present *Conure*, which could compact multiple (e.g., 6) tasks sequentially into a single deep UR model without network expansion and forgetting. *Conure* is conceptually simple, easy to implement, and applicable to a broad class of sequential encoder networks.
- We instantiate *Conure* by using temporal convolutional network (TCN) [43] as the backbone network for case study, and report important results for both TCN and the self-attention based network (i.e., Transformer [36]).
- We provide many useful insights regarding performance of various learning paradigms in the field of recommender systems and user modeling. We demonstrate that *Conure* largely exceeds its counterpart that performs the same continual learning process but without purposely preserving old knowledge. Moreover, *Conure* matches or exceeds separately trained models, typical transfer learning and multi-task learning approaches, which require either more model parameters or more training examples.

2 RELATED WORK

Our work intersects with research on user modeling (UM) and recommender systems (RS), transfer learning (TL), and continual learning (CL). We briefly review recent advances below.

2.1 User Modeling and Recommendation

User modeling refers to the process of obtaining the user profile, which is a conceptual understanding of the user. It is an important step towards personalized recommender systems. One common research line of UM is based on representation learning, where users or their behaviors are modeled and represented by certain types of machine learning algorithms [26, 38, 43, 46]. These well-trained digital user models are often called user representations.

Over recent years, deep neural networks have become dominant techniques for user representation learning. Among many, deep structured semantic models (DSSM) [14], deep or neural factorization machines (DeepFM/NFM) [9, 11] have become some representative work based on supervised representation learning. However, these learned UR models have been shown useful only for a specific task. One reason is that the supervised learning objective functions usually focus upon a specific goal only [42], which may not generalize well to other tasks.

Differently, PeterRec presented a self-supervised pretraining approach based on the sequential recommendation model NextItNet [43]. The pretraining process used is the prediction of the next user-item interaction in the user behavior sequence. By modeling

[2] Note recent work in [23, 30] also introduced a 'lifelong' learning solution for RS, the main difference between our paper and them is described in Section 2.3.

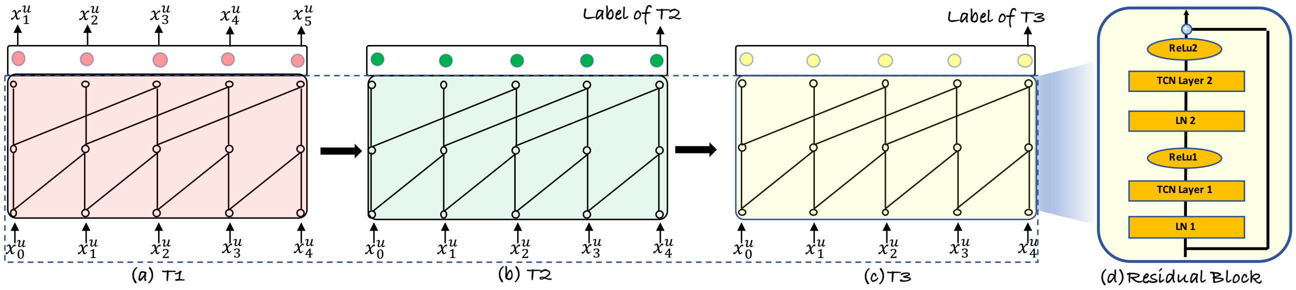


Figure 2: The continual learning framework with the TCN architecture without purposely preserving prior knowledge. (a)(b) and (c) show the network architectures of T_1, T_2, and T_3, respectively. The dashed frame denotes the backbone (i.e., the so-called user representation) encoder network, whose parameters are shared and will be modified (from red in T_1, green in T_2, to yellow in T_3) during training of each task. The top layers with red, green and yellow colors on (a) (b) and (c) respectively are task-specific prediction layers. (d) is a typical residual block of TCN.

the inherent relations of behavior sequences, the learned user representations become universal rather than specialized, and thereby can be used for many other tasks. Nevertheless, PeterRec enables only one-time TL between the first task (say, T_1) and the other task (e.g, T_2 or T_3), i.e., $T_1 \rightarrow T_2$, $T_1 \rightarrow T_3$, ..., $T_1 \rightarrow T_N$ rather than the continual TL among all tasks, e.g., $T_1 \rightarrow T_2 \rightarrow T_3, ..., \rightarrow T_N$ or $T_1 \rightarrow T_N \rightarrow T_3, ..., \rightarrow T_2$.[3] In this paper, we design and instantiate *Conure* based on PeterRec-style network architecture, which can be seen as an extension of PeterRec towards continual UR learning.

2.2 Transfer Learning

TL is typically based on a two-stage training paradigm: first pretraining a base model on the source dataset and then finetuning a new model on the target dataset with part or all of the pretrained parameters as initialization. Following PeterRec, we choose the well-known temporal (a.k.a. dilated) convolutional network (TCN) [41, 43] as the pretrained base model for case study given its linear complexity and superb performance in modeling sequences [2, 4, 27, 33, 37, 38, 45]. *Conure* is more related to the finetuning stage, which can be in general classified into the four types [6, 42]: i) finetuning only the softmax layer with the pretrained network as a feature extractor; ii) finetuning some higher layers while keeping the bottom layers frozen; iii) finetuning the entire pretrained model; and iv) finetuning only some newly added adaptor networks like PeterRec.

2.3 Continual Learning

CL refers to the continuous learning ability of an AI algorithm throughout its lifespan. It is regarded as an important step towards general machine intelligence [28]. While it has been explored in computer vision (CV) [8, 18, 21, 39, 44] and robot learning [24, 34], to our best knowledge, such task-level CL has never been studied for user modeling and recommender systems. In fact, it is largely unknown whether the learning paradigms, frameworks, and methodologies for other domains are useful or not to address our problem. Meanwhile, there are also some recent work in [23, 29, 30] claiming that RS models should have the so-called 'lifelong' learning capacity. However, their methodologies are designed only to model long-term user behaviors or new training data from the same distribution or task, which distinguish from *Conure*, capable of sequentially or separately learning very ***different*** tasks.

3 PRELIMINARIES

We begin with formulating the continual learning (CL) paradigm for user representations. Then, we perform experiments to verify the impacts of the catastrophic forgetting and the over-parameterization issues for deep user representation models.

3.1 Task Formulation

Suppose we are given a set of consecutive tasks $\mathbf{T} = \{T_1, T_2, ..., T_N\}$, where $\mathbf{T}$ is theoretically unbounded and allowed to increase new tasks throughout the lifespan of a CL algorithm. First we need to learn the base representations for users in T_1, and then ensure the continual learning of them if they appear in the following tasks, i.e., $\{T_2, ..., T_N\}$, so as to achieve more comprehensive representations. Denote $\mathcal{U}$ (of size $|\mathcal{U}|$) as the set of users in T_1. Each instance in T_1 contains a userID $u \in \mathcal{U}$, and his/her interaction sequence $\mathbf{x}^u = \{x_0^u, ..., x_n^u\}$ ($x_i^u \in \mathcal{X}$), i.e., $(u, \mathbf{x}^u) \in T_1$, where x_t^u is the t-th interaction of u and $\mathcal{X}$ (of size $|\mathcal{X}|$) is the set of items. For example, T_1 can be a video recommendation task where a number of user-video watching interactions are often available. Note that since in T_1 we are learning the base user representations, we assume that users in T_1 have at least several interactions for learning, although theoretically *Conure* works even with one interaction. On the other hand, each instance in $\{T_2, ..., T_N\}$ is formed of a userID $u \in \tilde{\mathcal{U}} \subseteq \mathcal{U}$ and a supervised label $y \in \mathcal{Y}$ (of size $|\mathcal{Y}|$), i.e., $(u, y) \in T_i$. If u has more than one label, say g, then there will be g instances for u. In our CL setting, $\{T_2, ..., T_N\}$ can be different tasks, including various profile (e.g., gender) prediction and item recommendation tasks, where y denotes a specific class (e.g., male or female) or an itemID, respectively. After training of $\mathbf{T}$, our *Conure* should be able to serve all tasks in $\mathbf{T}$ by one individual model.

3.2 Learning Sequential Tasks with TCN

In the training stage, *Conure* learns tasks in $\mathbf{T}$ one by one (e.g., $T_1 \rightarrow T_2 \rightarrow T_3, ..., \rightarrow T_N$) with only one backbone network, as shown in Figure 2. We present this vanilla CL procedure as follows.

[3] '→' denotes the direction of TL.

Training of T_1: As the first task, we should learn the base user representation (UR) which is expected to be universal rather than task-specific. To do so, we model the user interaction sequence $\mathbf{x}^u$ by an autoregressive (a.k.a. self-supervised) learning manner. Such training method was introduced into sequential recommender systems by NextItNet, which is also very popular in computer vision (CV) [35], natural language processing (NLP) [7, 36]. Formally, the joint distribution of a user sequence is represented as the product of conditional distributions over all user-item interactions:

$$p(\mathbf{x}^u; \Theta) = \prod_{j=1}^{n} p(x_j^u | x_0^u, .., x_{j-1}^u; \Theta) \tag{1}$$

where the value $p(x_j^u | x_0^u, ..., x_{j-1}^u)$ is the probability of the j-th interaction x_j^u conditioned on all its past interactions $\{x_0^u, ..., x_{j-1}^u\}$. Figure 2 (a) illustrates this conditioning scheme with TCN as the backbone network (described later). After training of T_1, the backbone (i.e., the so-called user representation model) could be transferred for many other tasks T_i ($i \geq 2$) according to the study in [42].

Training of T_i: The training of T_i ($i \geq 2$) is shown in Figure 2 (b) and (c). T_i is connected with T_1 by userID. For each instance (u, y) on T_i, we take the interaction sequence of u (in T_1) as input and feed it to its sequential encoder network, i.e., the backbone of T_1 as well. Let $\boldsymbol{E}_0 \in \mathbb{R}^{n \times f}$ be the embedding matrix of $\mathbf{x}^u$, where f is the embedding size. After passing it through the encoder network, we obtain the final hidden layer, denoted as $\boldsymbol{E} \in \mathbb{R}^{n \times f}$. Then, a dense prediction (or softmax) layer is placed on the last index vector of $\boldsymbol{E}$, denoted by $\boldsymbol{g}_{n-1} \in \mathbb{R}^f$. Finally, we can predict scores $\boldsymbol{h} \in \mathbb{R}^{|\mathcal{Y}|}$ with respect to all labels in $\mathcal{Y}$ by $\boldsymbol{h} = \boldsymbol{g}_{n-1}\boldsymbol{W} + \boldsymbol{b}$, where $\boldsymbol{W} \in \mathbb{R}^{f \times |\mathcal{Y}|}$ and $\boldsymbol{b} \in \mathbb{R}^{|\mathcal{Y}|}$ denote the projection matrix and bias term, respectively.

In terms of the training loss of T_i, one can apply either a ranking or a classification loss. In this paper, we report results using BPR [31] loss with the popular item-frequency[4] based negative sampling (see [40]) for top-N item recommendation tasks and the cross-entropy classification loss for profile prediction tasks.

Backbone Network: For better illustration, we instantiate *Conure* using the TCN architecture in the following despite that the framework is network-agnostic. Apart from the embedding layer, the TCN encoder is composed of a stack of temporal convolutional layers, every two of which are wrapped by a residual block structure, as shown in Figure 2 (d). The l-th residual block is formalized as

$$\boldsymbol{E}_l = \boldsymbol{F}_l(\boldsymbol{E}_{l-1}) + \boldsymbol{E}_{l-1} \tag{2}$$

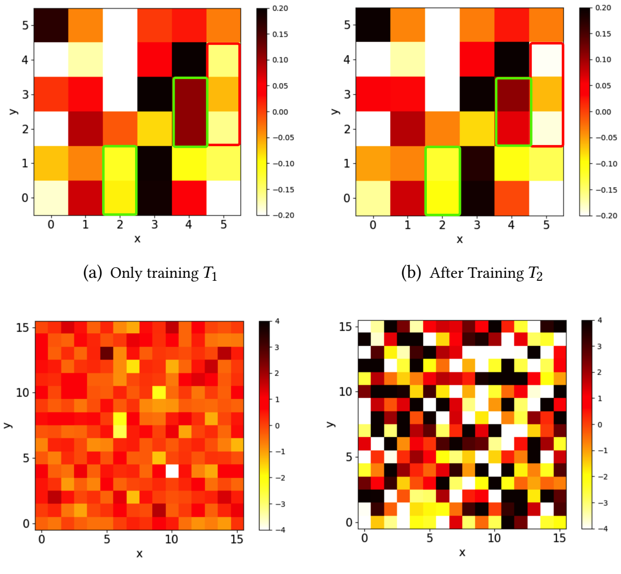
where $\boldsymbol{E}_{l-1}$ and $\boldsymbol{E}_l$ are the input and output of the l-th residual block respectively, and $\boldsymbol{F}$ is the residual mapping to be learned

$$\boldsymbol{F}_l(\boldsymbol{E}_{l-1}) = \sigma(\phi_2(LN_2(\sigma(\phi_1(LN_1(\boldsymbol{E}_{l-1})))))) \tag{3}$$

where σ is the ReLu [25] operation, LN is layer normalization [1] and ϕ is the TCN layer. Biases are omitted for simplifying notations.

3.3 Forgetting from T_1 to T_2

We investigate the catastrophic forgetting issue by sequentially learning T_1 and T_2. Since the model on T_2 shares the same backbone network as T_1, the optimization of it for T_2 will also lead to the parameter modification of T_1. We show the comparisons of weights and the final hidden vector $\boldsymbol{g}_{n-1}$ (of a randomly selected user-item interaction sequence) before and after the training of T_2 on Figure 3. Note in our experiments we use the convolution kernel $1 \times 3 \times 256$, where 1×3 and 256 are the kernel size and number of channels, respectively. To clearly demonstrate the difference, we select channels from the top 12-th indices, i.e., $1 \times 3 \times 12$ and depict them on (a) and (b).



(a) Only training T_1

(b) After Training T_2

(c) Only training T_1

(d) After training T_2

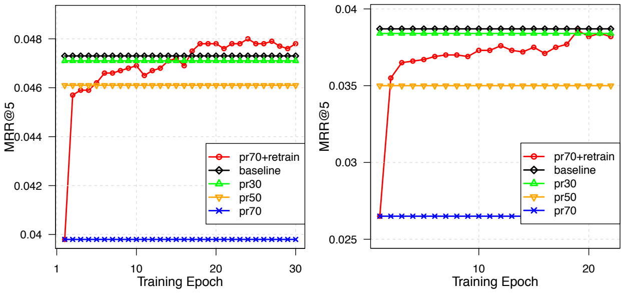
Figure 3: Forgetting issue during continual learning. (a) and (b) represent a reshaped 2-D (i.e., from $1 \times 3 \times 12$ to 6×6) convolution kernel of the last hidden layer, while (c) and (d) represent a reshaped (i.e., from 256 to 16×16) 2-D matrix of $\boldsymbol{g}_{n-1}$. Significantly different pixels on (a) (b) are marked by the red & green frames.

At first glance, (a) and (b) look quite similar. It seems that the forgetting issue is not as serious as we imagined. However, we notice that the prediction accuracy (mean reciprocal rank MRR@5) drops drastically from 0.0473 to 0.0010 (almost completely forgetting) when performing exactly the same evaluation on T_1 with the newly optimized parameters by training T_2. The degradation of performance implies more serious forgetting problem, compared with deep models in other domains [17, 18]. To identify the cause, we further check the changes of $\boldsymbol{g}_{n-1}$, which directly determines the final results together with the prediction layer of T_1. Clearly, the subfigure on (c) and (d) shows very big changes of $\boldsymbol{g}_{n-1}$ after learning T_2. In fact, we find that most of the weights on (a) have already been modified, but with a relatively small range (around ±20%), which is thus not very visible on figures. For instance, the first value with $(x, y) = (0, 0)$ changed from -0.1840 on (a) to -0.1645 on (b). It is reasonable that such small weight changes on many layers may incur cumulative effect, and lead to largely different outputs. i.e., the so-called catastrophic forgetting.

3.4 Over-parameterization phenomenon

We remove a certain percentage of unimportant/redundant parameters for each TCN layer trained after T_1. The importance of a

[4] Item-frequency based negative sampler has shown better performance than the random sampler in much literature w.r.t. the top-N metrics, such as MRR@N and NDCG@N [12]



(a) $f = 256$ with 16 CNN layers

(b) $f = 64$ with 8 CNN layers

Figure 4: Network trimming. “pr30", “pr50" & “pr70" denote pruning 30%, 50%, 70% parameters of the convolutional layers, respectively. “pr70+retrain" means performing retraining on the pruned network. Each epoch has $1000 * 32$ examples, where 32 is batch size.

parameter is measured by its absolute value sorted in the same layer. This process is often referred to as network trimming or pruning [13], which was originally invented for model compression [10, 22]. We report the pruning results in Figure 4. It shows that simply removing unimportant parameters results in a loss in accuracy — the more are pruned, the worse it performs. E.g, pruning 70% parameters hurts the accuracy seriously due to the sudden change in network connectivity. Fortunately, performing retraining on the pruned network (i.e., “pr70+retrain”) regains its original accuracy quickly, as shown on both (a) & (b). This, for the first time, evidences that over-parameterization or redundancy widely exists in the deep user representation model. Moreover, we note that even the network with a much smaller parameter size, i.e., having not reached its full ability, is still highly redundant, as shown on (b).

4 CONURE

Driven by the above insights, we could develop *Conure* for multiple tasks by exploiting parameter redundancy in deep user representation models: first removing unimportant parameters to free up space for the current task, then learning new tasks and filling task-specific parameters into the freed up capacity. To obtain positive transfer learning, important parameters from past tasks should be employed; to prevent forgetting, these important parameters should be kept fixed when learning new tasks. Figure 5 gives an overview of *Conure*.

Specifically, we begin by assuming that the base user representations have been obtained by training T_1 (see Figure 2 (a)). Before learning a new task (e.g., T_2), we first perform network pruning [10] to remain only a percentage of important parameters (light red cells in Figure 5 (a)) on the backbone network. After pruning, the model performance could be affected because of the big changes in network structure. We perform retraining (Figure 5 (b)) over these important parameters on the pruned architecture so as to regain its original performance, After this step, there are some free parameters left (the white cells in (b)), which are allowed to be optimized when learning a new task. In this way, when a new task arrives, *Conure* keeps learning it by only back propagating these free parameters while the remaining important parameters are hereafter kept fixed (Figure 5 (c)) for all future tasks. Next, by iteratively performing such network trimming (Figure 5 (d)) & retraining (Figure 5 (e)) on parameters of the current task, the user model could accommodate more tasks. Our idea here is initially inspired by [22] to some extent. The key difference is that unimportant/redundant parameters in [22] are replaced into important parameters from an external network for accuracy improvement, while unimportant parameters in *Conure* are re-optimized based on the new task so as to realize continual representation learning. We notice that similar attempts have been recently proven effective for solving problems in other research fields [7, 39]. In what follows, we provide a detailed explanation by specifying the TCN recommender as the backbone.

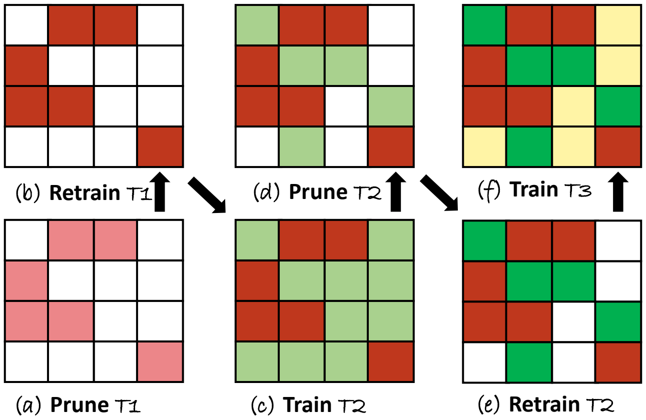


Figure 5: Overview of *Conure*. The subfigures from (a) to (f) denote parameter matrices of T_1, T_2, and T_3. The black arrows denotes the continual learning process. The white cells represent invalid parameters (i.e. set them to zero) for the current task, while the red (including both light red and dark red), green and yellow cells represent task-specific parameters for T_1, T_2, and T_3, respectively. Parameters of T_2 include both red and green cells, similarly, parameters of T_3 include all red, green and yellow cells.

4.1 Methodology Details

Redundancy Trimming. Parameters of *Conure* are mainly from the bottom embedding layer, middle layers, and the task-specific prediction layers. Both embedding and middle layers are allowed to be pruned. Despite that, we empirically find that the performance will not be affected even we keep parameters in the the embedding layer fixed after training T_1. This is very likely because lower-level features are more task-agnostic, similarly as in [42]. The middle layers of TCN consist of the temporal convolutional layers with bias terms, normalization and ReLu layers (see Figure 2 (d)). Given that the normalization layer and bias terms have very few parameters, we can keep them fixed for simplicity after training T_1.[5] Thereby, we conclude that to endow the continual learning capacity to *Conure*, one just needs to manipulate parameters of the hidden (convolutional) layers. This property is desirable as it helps *Conure* to reduce task-specific overhead in both computation and storage, and makes the learning process and parameters more manageable. Besides, we find that such property is also applicable to other types of networks, such as self-attention based Transformer [15] (see Section 5.6).

[5] Note that we place the normalization layer before the TCN layer, as shown in Figure 2 (d), otherwise, we strongly suggest optimizing it along with the TCN layer in the following tasks.

The pruning process of T_1 is illustrated in Figure 5 (a). To facilitate discussion, we describe it by using a convolutional layer. Formally, let $Z_{T_1} \in \mathbb{R}^{a \times b}$ be the weight of a convolutional layer, where $a \times b$ is the weight shape[6]. Assume we need to prune away Q_{T_1} (e.g., $Q_{T_1} = 70\%$) parameters on T_1. Before pruning, we rank all parameters (from the smallest to the largest) by a score function $g(Z_{T_1})$, where $g(Z_{T_1}^k) = |Z_{T_1}^k|$ in this paper.[7] Correspondingly, we obtain the threshold value δ with index $Q_{T_1} * h(Z_{T_1})$, where $h(Z_{T_1})$ is the number of parameters in Z_{T_1}. δ distinguishes the less important parameters from important ones. To realize pruning, we introduce a binary mask $G_{T_1} \in \{0, 1\}$ with the same shape as Z_{T_1}, defined by

$$G_{T_1}^k = \begin{cases} 1 & g(G_{T_1}^k) > \delta \\ 0 & g(G_{T_1}^k) < \delta \end{cases} \tag{4}$$

The effective weights after pruning becomes $Z_{T_1} \odot G_{T_1}$, where $\odot$ is element-wise product operator. This is reflected in Figure 5 (a), where white cells denote these trimmed redundant parameters, and their values are set to zero when performing convolution. Finally, these pruning masks G_{T_1} for all convolutional layers are saved for the next training stage.

Retraining. As shown in Figure 4, in the beginning *Conure* will experience a decline in performance by using the pruned structure, due to big changes in neural network structure. To regain its original performance, *Conure* performs retraining on the pruned architecture as demonstrated in Figure 5 (b). Due to the existence of G_{T_1}, only important parameters are re-optimized, while pruned parameters ($Z_{T_1} \odot (X - G_{T_1})$) whose values are set to zero keep unchanged because no gradients are created for them[8]. As shown from (a) to (b), parameters represented by the light red cells are modified to new ones with dark red colors, from $Z_{T_1} \odot G_{T_1}$ to $\hat{Z}_{T_1} \odot G_{T_1}$. We refer to $\hat{Z}_{T_1} \odot G_{T_1}$ as condensed parameters of T_1, which keep fixed at this point onwards. After a period of retraining, the performance on T_1 is very likely to recover as long as the pruning percentage is not too large. The updated parameters $\hat{Z}_{T_1}$ are saved to replace the original Z_{T_1} for the next stage.

The pruning and retraining operations on T_i ($i > 1$) will be executed only on task-specific parameters of T_i, where important parameters from T_1 to T_{i-1} are not allowed to be modified. For example, after training T_2, *Conure* once again performs pruning and retraining to prepare it for T_3. As shown in Figure 5 (d) and (e), only green cells from T_2 are pruned, while all red cells keep fixed. This allows *Conure* to always focus on optimization of the task at hand.

New task Training via knowledge retention. At this phase, *Conure* is required to accomplish two goals: i) achieving positive transfer on the new task T_i by leveraging condensed parameters (i.e., dark color cells in Figure 5) from T_1 to T_{i-1}; ii) overcoming forgetting these condensed parameters when learning T_i. To this end, we only allow the redundant parameters of T_i to be modified whereas condensed parameters from all past tasks are employed as prior knowledge and kept frozen only for forward propagation.

Table 1: Number of instances. The number of distinct items $|\mathcal{X}|$ in T_1 for TTL and ML is $646K$ and $54K$ ($K = 1000$), respectively. The number of labels $|\mathcal{Y}|$ is $18K$, $8K$, 8, 2, 6, respectively from T_2 to T_6 in TTL, and $26K$, $16K$, respectively from T_2 to T_3 in ML. $M = 1000K$.

Data	T_1	T_2	T_3	T_4	T_5	T_6
TTL	$1.47M$	$2.70M$	$0.27M$	$1.47M$	$1.47M$	$1.02M$
ML	$0.74M$	$3.06M$	$0.82M$	-	-	-

The weight used for learning T_i, i.e., Z_{T_i}, is given as:

$$Z_{T_i} = \underbrace{\hat{Z}_{T_{i-1}} \odot (X - \sum_{j=1}^{i-1} G_{T_j})}_{\text{freed up parameters}} + \underbrace{\text{stop\_gradient}(\hat{Z}_{T_{i-1}} \odot \sum_{j=1}^{i-1} G_{T_j})}_{\text{past condensed parameters}} \tag{5}$$

where $\hat{Z}_{T_{i-1}}$ is the weight of T_{i-1} after retraining, G_{T_j} is the task-specific weight mask generated by pruning, and stop_gradient is an operator that prevents the gradient from back propagation. For example, by performing training on T_2, the white cells are activated to light green, as shown in Figure 5 (c), while the dark red cells (condensed parameters) are kept unchanged. Following this way, *Conure* could perform iterative redundancy pruning and parameter retraining for new coming tasks so as to add more tasks into the backbone network. This process can be repeated until all tasks are added or no free capacity is available.

Overhead. In contrast to the sequential training described in section 3.2, *Conure* incurs additional storage overhead by maintaining the sparse mask G_{T_i}. However, as analyzed, if (after pruning) a parameter is useful for T_i , then it is used for all the following tasks $\{T_{i+1}, ..., T_N\}$, and meanwhile, it had actually been ignored for all the past tasks $\{T_1, ..., T_{i-1}\}$. This means the values corresponding to these parameters in the masks before and after T_i will be set as zero. Thus, the total number of additional non-zero (i.e., one) parameters in these sparse masks of all tasks in $\mathbf{T}$ is upper-bound to the size of the convolution parameters in the backbone network. Hence, *Conure* is much more parameter-efficient than the individually trained network for each task.

Inference. Once given a selected taskID, we can obtain the inference network of *Conure* which has the same structure as that developed for training for this task. Its only computation overhead is the masking operation which is implemented by multiplying convolution kernels with sparse tensors in an element-wise manner.

5 EXPERIMENTS

We assess the sequentially learned user representations by *Conure* on two tasks: personalized recommendations & profile predictions.

5.1 Experimental Settings

Datasets. As for the first work in continual UR learning over tasks, we find two public datasets to back up our key claim. They are the Tencent TL dataset released by PeterRec [42], referred to as TTL[9], and the movielens[10] dataset, referred to as ML. To be specific, TTL includes six different datasets connected by userID — three for item recommendations and three for profile classifications. Each instance

[6] Note the original convolutional kernel has a 3D shape, here we simply reshape it to 2D for better discussion. This process applies to weights with any shape or dimension.

[7] '||' denotes the symbol of absolute value.

[8] $X = ones(G)$ is a tensor with all elements one.

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

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

Table 2: Accuracy comparison. #B is the number of backbone networks. The left and right of '||' represent TTL and ML, respectively. *Conure-* denotes *Conure* that has not experienced the pruning operation after training on the current task. The worse and best results are marked by '▽' and '△', respectively.

Model	T_1	T_2	T_3	T_4	T_5	T_6	#B	T_1	T_2	T_3	#B
DNN	0.0104	0.0154	0.0231	0.7131	0.8908	0.6003	**6**	0.0276	0.0175	0.0313	**3**
SinMo	0.0473	0.0144	0.0161	0.7068	0.8998	0.5805▽	**6**	0.0637	0.0160	0.0259▽	**3**
SinMoAll	0.0009▽	0.0079 ▽	0.0124▽	0.5640▽	0.7314▽	0.6160	**1**	0.0038▽	0.0145▽	0.0310	**1**
FineSmax	0.0473	0.0160	0.0262	0.6798	0.8997	0.6070	**1**	0.0637	0.0150	0.0262	**1**
FineAll	0.0473	0.0172	0.0271	0.7160△	0.9053	0.6132	**6**	0.0637	0.0189	0.0325	**3**
PeterRec	0.0473	0.0173	0.0275	0.7137	0.9053	0.6156	**1**	0.0637	0.0182	0.0308	**1**
MTL	-	0.0151	0.0172	0.7094	0.8979	0.6027	**1**	-	0.0167	0.0276	**1**
Conure-	0.0473	0.0174	0.0286	0.7139	0.9051	0.6180	-	0.0637	0.0183	0.0347	-
Conure	0.0480△	0.0177△	0.0287△	0.7146	0.9068△	0.6185△	**1**	0.0656△	0.0197△	0.0353△	**1**

$(u, \mathbf{x}^u)$ in the dataset of T_1 contains a userID and his recent 100 news & video watching interactions on the *QQ Browser* platform; each instance (u, y) in T_2 contains a userID and one of his clicking (excluding thumbs-up) interactions on the *Kandian* platform; each instance in T_3 contains a userID and one of his thumb-up interactions on *Kandian*, where thumb-up represents more satisfactory than clicks. Each instance in T_4, T_5, T_6 contains a userID and his/her age, gender, and life status categories, respectively. We apply similar pre-processing for ML to mimic an expected CL setting, where each instance in T_1 contains a userID and his recent 30 clicking (excluding 4- and 5-star) interactions, each instance in T_2 contains a userID and an item that is rated higher than 4, and each instance in T_3 contains a userID and one of his 5-star items. A higher star means more satisfactory, the prediction of which is regarded as a harder task. Table 1 summarizes the dataset statistics.

Evaluation Protocols. To evaluate *Conure*, we randomly split each dataset in T_i into training (80%), validation (5%) and test (15%). We save parameters for each model only when they achieve the highest accuracy on the validation sets, and report results on their test sets. We use the popular top-N metric MRR@5 (Mean Reciprocal Rank) [40] to measure the recommendation tasks, and the classification accuracy (denoted by Acc, where Acc = number of correct predictions/total number of instances) to measure the profile prediction tasks.

Compared Methods. So far, there is no existing baseline for continual UR learning over different tasks. To back up our claim, we first present a typical two-layer DNN network following [5] for reference, where for learning T_i $(i > 1)$ the interaction sequence in T_1 is used as the outside features. Note we have omitted baselines such as DeepFM [9] and NFM [11] since in [42], authors showed that FineAll and PeterRec outperformed them. Except DNN, all of them apply the same TCN network architecture, shared hyper-parameters and sequential learning pipelines (except MTL trained simultaneously) for strict and meaningful comparisons.

- **SinMo**: Trains a single model for every task from scratch and applies no transfer learning between tasks. SinMo uses the same network architectures as *Conure* in each training stage (see Figure 2 (a) (b) and (c)) but is initialized randomly.
- **SinMoAll**: Applies a single backbone network for all tasks trained one by one without preserving parameters learned from previous tasks, as described in Section 3.2.
- **FineSmax**: After training T_1, only the final softmax layer for T_i $(i > 1)$ is finetuned, while all parameters from its backbone network are kept frozen & shared throughout all tasks.
- **FineAll**: After training T_1, all parameters for T_i are finetuned. To avoid the forgetting issue in SinMoAll, it requires to maintain additional storage for parameters of each task.
- **PeterRec**: Is a parameter-efficient transfer learning framework which needs to maintain only a small number of separate parameters for the model patches and softmax layers, while all other parameters are shared after T_1, see [42].
- **MTL**: Is a standard multi-task optimization via parameter sharing [3] in the backbone network. Since not all users have training labels in each task, we perform MTL only using two objectives, one is T_1 and the other is T_i $(i > 1)$.

Hyper-parameter. We assign the embedding & hidden dimensions f to 256 for all methods since further increasing yields no obvious accuracy gains. The learning rate is set to 0.001 for T_1 and 0.0001 for other tasks, similar to PeterRec. We use the Adam [16] optimizer in this paper. The regularization coefficient is set to 0.02 for all tasks except T_3 on TTL, where it is set to 0.05 for all models due to the overfitting problem. All models use dilation $4\times\{1, 2, 4, 8\}$ (16 layers) for TTL and $6\times\{1, 2, 4, 8\}$ (24 layers) for ML. The batch size b is set to 32 for T_1 and 512 for other tasks due to GPU memory consideration. Following PeterRec, we use the sampled softmax for T_1 with 20% sampling ratio. The popularity-based negative sampling coefficient is set to 0.3 (a default choice in [40]) for T_2 and T_3 for all models.

5.2 Performance Comparison and Insights

We perform sequential learning on the training sets of all tasks from T_1 to T_i ($i = 6$ & 3 for TTL and ML, respectively), and then evaluate them on their test sets. The pruning ratios of *Conure* are 70%, 80%, 90%, 80%, 90%, 90%, respectively from T_1 to T_6 on TTL (with 32%

free parameters left), and 70%, 80%, 70%, respectively from T_1 to T_3 on ML (with 39% free parameters left). We report results in Table 2.

Catastrophic forgetting: We observe that SinMoAll performs the worst among all tasks except the last one (i.e., T_6 of TTL, and T_3 of ML). It is even much worse than SinMo which has no transfer learning between tasks. This is because SinMoAll uses one backbone network for all tasks, suffering from severe catastrophic forgetting for its past tasks — i.e., after learning T_i, most parameters for T_1 to T_{i-1} are largely modified, and therefore it cannot make accurate prediction for them anymore. Nevertheless, it yields relatively good results on T_6 as there is no forgetting for the last task. In contrast, *Conure* clearly exceeds SinMoAll by overcoming forgetting although it also employs only one backbone network.

One-time TL from T_1 to T_i (e.g., $T_1 \rightarrow T_2$, $T_1 \rightarrow T_3$, or $T_1 \rightarrow T_6$): SinMo shows worse results (after T_1) comparing to other baselines because of no transfer learning between tasks. By contrast, FineAll produces much better results, although the two models share exactly the same network architecture and hyper-parameters. The main advantage of FineAll is that before training each T_i ($i \geq 2$), it has already obtained a well-initialized representation by training T_1.

Conure and PeterRec perform competitively with FineAll on many tasks, showing their capacities in doing positive transfer learning from T_1 to other tasks. But compared to FineAll, *Conure* and PeterRec are parameter very efficient since only one backbone network is applied for all tasks. In addition, FineAll largely surpasses FineSmax, indicating that only optimizing the final prediction/softmax layer is not expressive enough for learning a new task.

Multiple TL from T_1 to T_i (e.g., $T_1 \rightarrow \mathbf{T_2 \rightarrow T_3}, ..., \rightarrow T_6$): Compared to FineAll and PeterRec, *Conure-* yields around 4% and 7% accuracy gains on T_3 of TTL and ML, respectively. The better results are mainly from the positive transfer from T_2 to T_3, which cannot be achieved by any other model. To our best knowledge, so far *Conure* is the only model which could keep positive transfer learning amongst three or more tasks. Another finding is that *Conure-* does not obviously beat PeterRec and FineAll on T_4, T_5 and T_6. We believe that this is reasonable since there might be no further positive transfer from T_2, T_3 to T_4, T_5, T_6 given that it has experienced one-time effective transfer from T_1 to T_4, T_5, T_6.[11] But the good point is that *Conure* does not become worse even when there is no effective positive transfer. The slightly improved result of *Conure-* on T_6 mainly comes from its robustness since parameters of irrelevant tasks may act as good regularization to resist overfitting. By comparing *Conure-* and *Conure*, we find that properly pruning with retraining usually brings a certain percentage of improvements for deep user representation models.

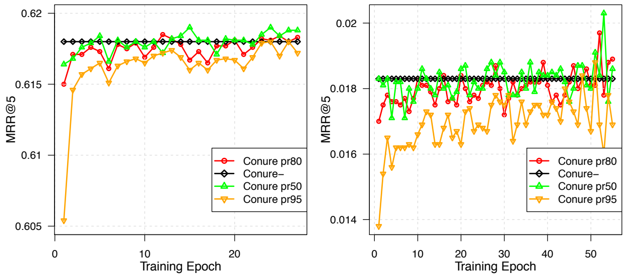
Performance of other baselines: MTL outperforms SinMo, showing the effects by applying multi-objective learning since the only difference between them is an additional T_1 loss in MTL. But it still performs worse than FineAll, PeterRec and *Conure*. One key weakness of MTL is that it has to consider the accuracy for all (i.e., 2) objectives simultaneously, and thus might not be always optimal for each of them. Besides, MTL is unable to leverage all training data since some users of T_1 have no training instances on T_i ($i \geq 2$). Meanwhile, the standard DNN performs relatively well on some tasks but much worse on T_1 because it is unable to model the sequential patterns in user actions. Another drawback is that such models (including DeepFM and NFM) have to be trained individually for each task, which are parameter-inefficient as well.

Table 3: Impact of T_2 on T_3. *Conure_noT_2* denotes training *Conure* on T_3 after T_1. *Conure_noT_2* and *Conure* both are the *Conure-* versions. TTL20% and ML20% denote the 20/80 train/test split.

	TTL	TTL20%	ML	ML20%
Conure_noT_2	0.0277	0.0245	0.0334	0.0295
Conure	0.0286	0.0261	0.0347	0.0309
Impro.	3.2%	6.5%	3.9%	4.7%

Table 4: Impact of task orders. Order1 is the original order as mentioned in Section 5.1. KC, KT and Life denotes the clicking dataset, the thumbs-up dataset and the life status dataset of *Kandian*, respectively. Results on T_1 are omitted due to the same accuracy. The left and right of '||' are results of *Conure-* and *Conure*, respectively.

Orders	KC	KT	Life	KC	KT	Life
Order1	0.0174	0.0286	0.6180	0.0177	0.0287	0.6185
Order2	0.0174	0.0289	0.6154	0.0177	0.0290	0.6152
Order3	0.0174	0.0289	0.6145	0.0177	0.0287	0.6149



(a) TTL T_6 (one epoch: 1000*b)

(b) ML T_2 (one epoch: 2000*b)

Figure 6: Impact of pruning percentages. The numbers (i.e., 50, 80, 95) denote pruning ratios.

5.3 Impact of T_2 for T_3

We perform more strict studies to verify the positive transfer from T_2 to T_3, since it is the unique ability of *Conure* distinguishing from all other models. To do so, we evaluate *Conure* on T_3 without training T_2 in advance. To clearly see the transfer effect, we also report results with a new split with 20% data for training and the remaining for testing since TL may not be necessary if there is enough task-specific data. As shown in Table 3, *Conure* clearly outperforms *Conure_noT_2* on both TTL and ML (with statistical significance). Particularly, *Conure* obtains 6.5% accuracy gain on TTL20% by learning T_2 ahead. Such findings well back up our claim and motivation regarding the advantage of *Conure* — i.e., it is particularly expert at the sequential tasks learning once they have a certain relatedness.

[11] Note it is not easy to find an ideal publicly available dataset, where all tasks share expected similarities. But in practice, there indeed exist many related tasks for both recommendation and profile prediction. For example, a recommender system may require to predict user various interactions, such as clicks, likes, comments, shares, follows and reposts, etc; similarly, a profile predictor may estimate user's profession, education and salary, which could potentially have some high correlations.

Table 5: Pruning and retraining both the embedding & convolutional layers. The left & right of '||' are tasks on TTL & ML.

Models	T_1	T_2	T_3	T_1	T_2	T_3
Conure-	0.0473	0.0175	0.0290	0.0637	0.0191	0.0341
Conure	0.0474	0.0177	0.0295	0.0645	0.0196	0.0347

Table 6: Results by specifying *Conure* with Transformer as the backbone network. The left and right of '||' represent tasks on TTL and ML, respectively. 'Mo' , 'FA', 'C-', 'C', denotes Models, FineAll, Conure- and Conure, respectively.

Mo	T_1	T_2	T_3	#B	T_1	T_2	T_3	#B
FA	0.0510	0.0161	0.0243	**3**	0.0654	0.0193	0.0321	**3**
C-	0.0510	0.0177	0.0288	-	0.0654	0.0198	0.0345	-
C	0.0513	0.0179	0.0289	**1**	0.0662	0.0200	0.0357	**1**

5.4 Impact of Task Orders

In this subsection, we are interested in investigating whether *Conure* is sensitive to the task orders. It is worth noting that the first task should not be changed since its responsibility is to obtain the base user representation. To be specific, we compare *Conure* with another two orders on TTL, namely $T_1 \rightarrow T_2 \rightarrow T_6 \rightarrow T_3$ (denoted as Order2) and $T_1 \rightarrow T_6 \rightarrow T_2 \rightarrow T_3$ (denoted as Order3). As shown in Table 4, *Conure* is in general not sensitive to task orders. Interestingly, *Conure* performs better on Life when it is trained lastly (i.e., Order1). One reason may be that parameters of previous tasks could also work as good regularization terms, which increase model robustness to noisy labels and overfitting. Its accuracy on Life with Order2 & 3 are almost the same as PeterRec in Table 2, which is further evidence for this argument. Likewise, *Conure-* performs slightly better on KT with order2 & 3, because KT is trained lastly.

5.5 Impact of Weight Pruning

In this subsection, we examine the impact of pruning. We plot the retraining processes of T_6 (on TTL) and T_2 (on ML) in Figure 6. As shown, a few epochs of retraining after pruning can recover the performance of *Conure-*. In particular, *Conure* is able to outperform *Conure-* even pruning away over 50% redundant parameters. For example, *Conure* improves MRR@5 of *Conure-* from 0.0183 to 0.0203 when pruning 50% parameters on T_2. In addition, we also notice that pruning too much percentage (e.g., 95%) of parameters could lead to worse performance or slower convergence, as shown on (b). In practice, we suggest tuning the pruning ratios from 50% to 80%.

Though *Conure* performs very well by performing continual learning on only middle layers, we hope to verify its applicability to the embedding layer. To this end, we prune and retrain both the embedding and convolutional layers. The pruning ratios for hidden layers remain the same as in Section 5.2, while for the embedding layer they are 30%, 80%, 80% for T_1, T_2 and T_3, respectively. As shown in Table 5, we observe that pruning and retraining additional parameters of the embedding layers reach similar results as in Table 2. An advantage by pruning the embedding layer is that more free capacity can be released to promote the future task learning.

5.6 Adaptability

Here we investigate whether the framework can be applied to other types of backbone networks. Inspired by the huge success of self-attention or Transformer in recent literature [6, 36], we specify *Conure* with the Transformer architecture as the encoder. We choose one attention head and two self-attention residual blocks due to its good performance in the validation set. Other hyper-parameters and setups are kept exactly the same as in Section 5.1. We prune and retrain only the linear transformation layers in the residual block (including weights from both the self-attention and feed-forward blocks). We report the results in Table 6. As shown, we basically achieve similar conclusions as before. Specifically, (i) compared with FineAll, *Conure* obtains obvious improvement on T_3 on both TTL and ML, since FineAll could only enable one-time transfer learning, e.g., from T_1 to T_3, but *Conure* could keep continual transfer learning from T_1, T_2 to T_3. (ii) *Conure* requires only one backbone network for three tasks whereas FineAll requires three to avoid forgetting. In addition, we also find that in contrast to TCN, *Conure* with Transformer as the backbone network usually yields some better results (see Table 2). But it is also worth noting Transformer requires quadratic time complexity to compute self-attention, whereas TCN has only linear complexity, which is much faster than Transformer when handling long-range interaction sequences.

6 CONCLUSIONS AND IMPACTS

In this paper, we have confirmed two valuable facts: i) better user representations could be learned in a sequential manner by acquiring new capacities and remembering old ones; ii) continually learned user representations can be used to solve various user-related tasks, such as personalized recommender systems and profile predictions. We proposed *Conure* — the first task-level lifelong user representation model, which is conceptually very simple, easy to implement, and requires no *very* specialized network structures. Besides, *Conure* has achieved comparable or better performance in contrast to the classic learning paradigms (including single-task, multi-task and transfer learning) with minimal storage overhead. We believe *Conure* has made a valuable contribution in exploring the lifelong learning paradigm for user representations, approaching the goal of one person, one model, one world.

For future work, there is still much room for improvement of *Conure* towards a more intelligent lifelong learner. First, while *Conure* is able to achieve positive transfer for new tasks, it could not in turn transfer the newly learned knowledge to improve old tasks. Second, while *Conure* can easily handle sequential learning for over six tasks, it is yet not a never-ending learner since it cannot automatically grow its architecture. Third, it is unknown whether the performance of *Conure* will be affected if there are contradictory tasks requiring optimization in opposite directions. We hope *Conure* would inspire new research work to meet these challenges. We also expect some high-quality real-world benchmark datasets could be released so as to facilitate research in this difficult area.

REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[2] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271* (2018).

[3] Rich Caruana. 1997. Multitask learning. *Machine learning* 28, 1 (1997), 41–75.

[4] Lei Chen, Fajie Yuan, Jiaxi Yang, Xiang Ao, Chengming Li, and Min Yang. 2021. A User-Adaptive Layer Selection Framework for Very Deep Sequential Recommender Models. (2021).

[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] Binzong Geng, Min Yang, Fajie Yuan, Shupeng Wang, Xiang Ao, and Ruifeng Xu. 2021. Iterative Network Pruning with Uncertainty Regularization forLifelong Sentiment Classification. In *Proceedings of the 44th International ACM SIGIR conference on Research and Development in Information Retrieval.*

[8] Siavash Golkar, Michael Kagan, and Kyunghyun Cho. 2019. Continual learning via neural pruning. *arXiv preprint arXiv:1903.04476* (2019).

[9] 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).

[10] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[11] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 355–364.

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

[13] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. 2016. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250* (2016).

[14] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 2333–2338.

[15] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.

[16] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[17] 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* (2017).

[18] Zhizhong Li and Derek Hoiem. 2017. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence* 40, 12 (2017), 2935–2947.

[19] 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*. 1930–1939.

[20] 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.

[21] 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.

[22] Fanxu Meng, Hao Cheng, Ke Li, Zhixin Xu, Rongrong Ji, Xing Sun, and Guangming Lu. 2020. Filter grafting for deep neural networks. In *CVPR*. 6599–6607.

[23] Fei Mi, Xiaoyu Lin, and Boi Faltings. 2020. Ader: Adaptively distilled exemplar replay towards continual learning for session-based recommendation. In *Fourteenth ACM Conference on Recommender Systems*. 408–413.

[24] Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Bishan Yang, Justin Betteridge, Andrew Carlson, Bhavana Dalvi, Matt Gardner, Bryan Kisiel, et al. 2018. Never-ending learning. *Commun. ACM* 61, 5 (2018), 103–115.

[25] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*. 807–814.

[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. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 596–605.

[27] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).

[28] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* 113 (2019), 54–71.

[29] Pi Qi, Xiaoqiang Zhu, Guorui Zhou, Yujing Zhang, Zhe Wang, Lejian Ren, Ying Fan, and Kun Gai. 2020. Search-based User Interest Modeling with Lifelong Sequential Behavior Data for Click-Through Rate Prediction. *arXiv preprint arXiv:2006.05639* (2020).

[30] Kan Ren, Jiarui Qin, Yuchen Fang, Weinan Zhang, Lei Zheng, Weijie Bian, Guorui Zhou, Jian Xu, Yong Yu, Xiaoqiang Zhu, et al. 2019. Lifelong Sequential Modeling with Personalized Memorization for User Response Prediction. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 565–574.

[31] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. 452–461.

[32] Yang Sun, Fajie Yuan, Ming Yang, Guoao Wei, Zhou Zhao, and Duo Liu. 2020. A Generic Network Compression Framework for Sequential Recommender Systems. *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining* (2020).

[33] Md Mehrab Tanjim, Hammad A Ayyubi, and Garrison W Cottrell. 2020. DynamicRec: A Dynamic Convolutional Network for Next Item Recommendation. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2237–2240.

[34] Sebastian Thrun and Tom M Mitchell. 1995. Lifelong robot learning. *Robotics and autonomous systems* 15, 1-2 (1995), 25–46.

[35] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. 2016. Pixel Recurrent Neural Networks. In *International Conference on Machine Learning*. 1747–1756.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[37] Jingyi Wang, Qiang Liu, Zhaocheng Liu, and Shu Wu. 2019. Towards Accurate and Interpretable Sequential Prediction: A CNN & Attention-Based Feature Extractor. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1703–1712.

[38] Jiachun Wang, Fajie Yuan, Jian Chen, Qingyao Wu, Chengmin Li, Min Yang, Yang Sun, and Guoxiao Zhang. 2021. StackRec: Efficient Training of Very Deep Sequential Recommender Models by Iterative Stacking. *Proceedings of the 44th International ACM SIGIR conference on Research and Development in Information Retrieval* (2021).

[39] Zifeng Wang, Tong Jian, Kaushik Chowdhury, Yanzhi Wang, Jennifer Dy, and Stratis Ioannidis. 2020. Learn-Prune-Share for Lifelong Learning. *arXiv preprint arXiv:2012.06956* (2020).

[40] Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. 2016. Lambdafm: learning optimal ranking with factorization machines using lambda surrogates. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 227–236.

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

[42] Fajie Yuan, Xiangnan He, Alexandros Karatzoglou, and Liguang Zhang. 2020. Parameter-Efficient Transfer from Sequential Behaviors for User Modeling and Recommendation. *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval* (2020).

[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*. ACM, 582–590.

[44] Friedemann Zenke, Ben Poole, and Surya Ganguli. 2017. Continual learning through synaptic intelligence. *Proceedings of machine learning research* 70 (2017), 3987.

[45] Pengyu Zhao, Kecheng Xiao, Yuanxing Zhang, Kaigui Bian, and Wei Yan. 2020. Amer: Automatic behavior modeling and interaction exploration in recommender system. *arXiv preprint arXiv:2006.05933* (2020).

[46] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. 2020. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In *CIKM*. 1893–1902.