Parameter-Efficient Transfer from Sequential Behaviors for User Modeling and Recommendation

Fajie Yuan
Tencent
Shenzhen, China
fajieyuan@tencent.com

Xiangnan He
University of Science and Technology of China
Hefei, China
xiangnanhe@gmail.com

Alexandros Karatzoglou*
Google
London, UK
alexandros.karatzoglou@gmail.com

Liguang Zhang
Tencent
Shenzhen, China
kanongzhang@tencent.com

ABSTRACT

Inductive transfer learning has had a big impact on computer vision and NLP domains but has not been used in the area of recommender systems. Even though there has been a large body of research on generating recommendations based on modeling user-item interaction sequences, few of them attempt to represent and transfer these models for serving downstream tasks where only limited data exists.

In this paper, we delve on the task of effectively learning a single user representation that can be applied to a diversity of tasks, from cross-domain recommendations to user profile predictions. Fine-tuning a large pre-trained network and adapting it to downstream tasks is an effective way to solve such tasks. However, fine-tuning is parameter inefficient considering that an entire model needs to be re-trained for every new task. To overcome this issue, we develop a parameter-efficient transfer learning architecture, termed as PeterRec, which can be configured on-the-fly to various downstream tasks. Specifically, PeterRec allows the pre-trained parameters to remain unaltered during fine-tuning by injecting a series of re-learned neural networks, which are small but as expressive as learning the entire network. We perform extensive experimental ablation to show the effectiveness of the learned user representation in five downstream tasks. Moreover, we show that PeterRec performs efficient transfer learning in multiple domains, where it achieves comparable or sometimes better performance relative to fine-tuning the entire model parameters.

1 INTRODUCTION

The last 10 years have seen the ever increasing use of social media platforms and e-commerce systems, such as Tiktok, Amazon or Netflix. Massive amounts of clicking & purchase interactions, and other user feedback are created explicitly or implicitly in such systems. For example, regular users on Tiktok may watch hundreds to thousands of micro-videos per week given that the average playing time of each video is less than 20 seconds [36]. A large body of research has clearly shown that these interaction sequences can be used to model the item preferences of the users [8, 18, 31, 35–37]. Deep neural network models, such as GRURec [14] and NextItNet [37], have achieved remarkable results in modeling sequential user-item interactions and generating personalized recommendations. However, most of the past work has been focused on the task of recommending items on the same platform, from where the data came from. Few of these methods exploit this data to learn a universal user representation that could then be used for a different downstream tasks, such as for instance the cold-start user problem on a different recommendation platform or the prediction of a user profile.

In this work, we deal with the task of adapting a singe user representation model for multiple downstream tasks. In particular, we attempt to use deep neural network models, pre-trained in an unsupervised (self-supervised) manner on a source domain with rich sequential user-item interactions, for a variety of tasks on target domains, where users are cold or new. To do so, we need to tackle the following issues: (1) construct a highly effective and general pre-training model that is capable of modeling and representing very long-range user-item interaction sequences without supervision. (2) develop a fine-tuning architecture that can transfer pre-trained user representations to downstream tasks. Existing recommender systems literature is unclear on whether unsupervised learned user representations are useful in different domains where the same users are involved but where they have little supervised labeled data. (3) introduce an adaptation method that enables the fine-tuning architecture to share most of the parameters across all tasks. Although fine-tuning a separate model for each task often performs better, we believe there are important reasons for reusing parameters between tasks. Taking scenarios on end user devices as an example, applying several different neural networks for each task with the same input is computationally expensive and memory intensive, due to the limited energy and storage resources [22, 29]. Even for the large-scale web applications, practitioners need to avoid maintaining a separate large model for every user [29], especially when there are a large number of tasks.

For tackling the third issue, two transfer techniques have been commonly used: [34]: (1) fine-tuning an additional output layer to project transferred knowledge from a source domain to a target domain, and (2) fine-tuning the last (few) hidden layers along with the output layer — which is said to be capable of sharing some low and mid-level parameters in the early layers of neural network models [34]. However, we find that fine-tuning only the output layer usually performs poorly in the recommendation scenario;

*This work was mostly done when Alexandros was at Telefonica Research, Spain.

fine-tuning the last few layers properly sometimes offers comparable performance compared to fine-tuning the entire network, but requires much manual effort since the number of layers to be tuned highly depends on the pre-trained model and target task. Thus far, there is no consensus on how to choose the number, which in practice often relies on an inefficient hyper-parameter search. In addition, fine-tuning the last few layers does not realize our goal to share most parameters of the pre-trained model. Moreover, revising the pre-trained model for every new task is inflexible for deploying and managing online systems.

To achieve the first two goals, we propose a two-stage training procedure. First, in order to learn a universal user representation, we employ sequential neural networks as our pre-trained model and train them with users' historical clicking or purchase sequences. Sequential models can be trained without manually labeled data using self-supervision which is essentially trained by predicting the next item on the sequence. Moreover sequential data is much easier to collect from online systems. In this paper, we choose NextItNet-style [36, 37] neural networks as the base models considering that they achieve state-of-the-art performance when modeling very long-range sequential user-item interactions [32]. Subsequently, we adapt the pre-trained model to downstream target tasks using supervised objectives. By doing so, we obtain an NLP [25] or computer vision (CV) [34]-like transfer learning framework.

To achieve the third goal that enables a high degree of parameter sharing for fine-tuning models between domains, we borrow an idea from the learning-to-learn method, analogous to learnet in [3]. The core idea of learning-to-learn is that the parameters of deep neural networks can be predicted from another [3, 26]; moreover, [5] demonstrated that it is possible to predict more than 95% parameters of a network in a layer given the remaining 5%. Motivated by these works, we are interested in exploring whether these findings hold for the transfer learning tasks in the recommender system (RS) domain. In addition, unlike above works, we are more interested in exploring the idea of parameter adaptation rather than prediction. Specifically, we propose a separate neural network, termed as model patch, which adapts the parameters of each convolutional layer in the pre-trained model to a target task. Each model patch consists of less than 10% of the parameters of the original convolutional layer. By inserting such model patches into the pre-trained models, our fine-tuning networks are not only able to keep all pre-trained parameters unchanged, but also successfully induce them for problems of downstream tasks without a significant drop in performance. We name the proposed model PeterRec, where 'Peter' stands for parameter efficient transfer learning.

The contributions of this paper are listed as follows:

- We propose a universal user representational learning architecture, a method that can be used to achieve NLP or CV-like transfer learning for various downstream tasks. More importantly, we are the first to demonstrate that self-supervised learned user representations can be used to infer user profiles, such as for instance the gender, age, preferences and life status (e.g., single, married or parenting). It is conceivable that the inferred user profiles by PeterRec can help improve the quality of many public and commercial services, but also raises concerns of privacy protection.
- We propose a simple yet very effective model patch network, which allows pre-trained weights to remain unaltered and shared for various downstream tasks.
- We propose two alternative ways to inject the model patches into pre-trained models, namely serial and parallel insertion.
- We perform extensive ablation analysis on five different tasks during fine-tuning, and report many interesting findings, which could be directions for future research in the RS domain.
- We will release our code and several high-quality datasets in this paper. To our best knowledge, this is the first large-scale recommendation datasets that can be used for both transfer & multi-domain learning. We hope our results and datasets can provide a benchmark to facilitate the research of transfer learning in the RS domain.

2 RELATED WORK

PeterRec tackles two research questions: (1) training an effective and efficient base model, and (2) transferring the learned user representations from the base model to downstream tasks with a high degree of parameter sharing. Since we choose the sequential recommendation models to perform this upstream task, we briefly review related literature. Then we recapitalize work in transfer learning and user representation adaptation.

2.1 Sequential Recommendation Models

A sequential recommendation (SR) model takes in a sequence (session) of user-item interactions, and taking sequentially each item of the sequence as input aims to predict the next one(s) that the user likes. SR have demonstrated obvious accuracy gains compared to traditional content or context-based recommendations when modeling users sequential actions [18]. Another merit of SR is that sequential models do not necessarily require user profile information and supervised labels, since user representations can be implicitly reflected by their past sequential behaviors by an autoregressive [37] training approach or a masked objective [36]. Amongst these models, researchers have paid special attention to three lines of work: RNN-based [14], CNN-based [31, 36, 37], and pure attention-based [18] sequential models. Generally speaking, standard RNN-based models rely on strict sequential dependencies during training, and thus, cannot fully take advantage of modern computing architectures, such as GPUs or TPU [37]. CNN and attention-based recommendation models do not have such a problems since the entire sequence can be observed during training and thus can be fully parallel. In addition, CNN-based models also yield better recommendation accuracy since much more neural layers can be stacked than RNNs. One well-known obstacle that prevents CNN from being a strong sequential model is the limited receptive field due to its small kernel size (e.g., 3×3). This issue has been cleverly approached by introducing the dilated convolutional operation, which enables an exponentially increased receptive field with unchanged kernel [36, 37]. By contrast, self-attention based sequential models, such as SASRec [18] may have time complexity and memory issues since they grow quadratically with the sequence length. Particularly, in our scenario, considering that most users have hundreds or even thousands of news and micro-video

watching behaviors in practice, SASRec is not an efficient choice. Therefore, we choose dilated convolution-based sequential neural network to build the pre-trained models by investigating both causal (i.e., NextItNet [37]) and non-causal (i.e., the bidirectional encoder of GRec [36]) convolutions in this paper.

2.2 Transfer Learning & Domain Adaptation

Transfer learning (TL) has recently become a research hotspot in many application fields of machine learning [6, 25, 34]. TL refers to methods that exploit knowledge gained in a source domain where a vast amount of training data is available, to improve a different but related problem in a target domain where only little labeled data can be obtained. Unlike much early work that concentrated on shallow classifiers (or predictors), e.g., matrix factorization in recommender systems [40], recent TL research has shifted to using large & deep neural network as classifiers, which has yielded significantly better accuracy [4, 17, 24, 39]. In fields like computer vision, transferred super deep neural networks have even achieved human-level classification accuracy [7]. However, this also brought up new challenges: (1) how to perform efficient transfer learning for end use applications, such as smart phones or consumer-oriented tablet devices, with limited computation and storage resources? (2) how to avoid overfitting problems for large neural network models when training examples are scarce in the target domain? To our knowledge, these types of research have not been well explored in the existing recommendation literature. In fact, we are even not sure whether it is possible to learn an effective user representation by *only* using their past behaviors (i.e., no user profiles & no other item features), and whether such representations can be transferred to improve the downstream tasks, analogous to learned visual features from ImageNet [10, 20] or embedding features from text corpus [6, 15, 25].

Closely related to this work, [24] recently introduced a DUPN model, which represents deep user perception network. DUPN is also capable of learning general user representations for multi-task purpose. But we find there are several key differences from this work. First, DUPN has to be pre-trained by a multi-task learning objective, i.e., more than one training loss. It showed that the learned user representations performed much worse if there are no auxiliary losses and data. By contrast, PeterRec is pre-trained by one single loss but can be adapted to multiple domains or tasks. To this end, we define the task in this paper as a multi-domain learning problem [27], which distinguishes from the multi-task learning in DUPN. Second, DUPN performs pre-training by relying on many additional features, such as user profiles and item features. It requires expensive human efforts in feature engineering, and it is also unclear whether the user representation work or not without these features. In fact, we find the way of pre-training in DUPN is very weak since it does not explicitly model the item dependencies in the user sequence. Similar issues have already been analyzed by authors in NextitNet when they compare with Caser [31]. Third, DUPN does not consider efficient transfer learning issue since it only investigates fine-tuning all pre-trained parameters and the final classification layer. By contrast, PeterRec fine-tunes a small fraction of injected parameters, but obtains comparable or better results than fine-tuning all parameters, and performs notably better than fine-tuning only the classification layer.

CoNet [16] is another cross-domain recommendation model using neural networks as the base model. To enable knowledge transfer, CoNet jointly trains two objective functions, among which one represents the source network and the other the target. One interesting conclusion was made by the authors of CoNet is that the pre-training and fine-tuning paradigm in their paper does not work well according to the empirical observations: the way of pre-training a multilayer perceptron network on the source domain and then transferring user representations to the target domain for fine-tuning does not yield performance improvements [16]. The negative results in CoNet may come from many aspects, such as the way of pre-training, the expressiveness of their user representations, as well as the quality of the pre-training data. In fact, neither CoNet nor DUPN provides evidence that fine-tuning with a pre-trained network performs better than fine-tuning from scratch, which, beyond doubt, is the fundamental assumption for TL in recommender systems. However, in this paper, we clearly demonstrate that our PeterRec notably improves the accuracy of downstream recommendation tasks by fine-tuning on the pre-trained model relative to training from scratch.

3 PETERREC

The training procedure of PeterRec consists of two stages. The first stage is learning a high-capacity user representation model on datasets with plenty of user sequential user-item interactions. This is followed by a supervised fine-tuning stage, where the pre-trained representation is adapted to the downstream task with explicit labels. In particular, we attempt to share the majority of parameters.

3.1 Notation

We begin with some basic notations. Suppose that we are given two domains: a source domain $\mathcal{S}$ and target domain $\mathcal{T}$. For example, $\mathcal{S}$ can be news or video recommendation where a large number of user interactions are often available, and $\mathcal{T}$ can be a different prediction task where user labels are usually very limited. In more detail, a user label in this paper can be an item he prefers in $\mathcal{T}$, an age bracket he belongs to, or the marital status he is in. Let $\mathcal{U}$ (of size $|\mathcal{U}|$) be the set of users shared in both domains. Each instance in $\mathcal{S}$ (of size $|\mathcal{S}|$) consists of a userID $u \in \mathcal{U}$, and the unsupervised interaction sequence $\mathbf{x}^u = \{x_1^u, ..., x_n^u\}$ ($x_i^u \in \mathcal{X}$), i.e., $(u, \mathbf{x}^u) \in \mathcal{S}$, where x_t^u denotes the t-th interacted item of u and $\mathcal{X}$ (of size $|\mathcal{X}|$) is the set of items in $\mathcal{S}$. Correspondingly, each instance in $\mathcal{T}$ (of size $|\mathcal{T}|$) consists of a userID u, along with the supervised label $y \in \mathcal{Y}$, i.e., $(u, y) \in \mathcal{T}$. Note if u has g different labels, then there will be g instances for u in $\mathcal{T}$.

We also show the parameters in the pre-trained and fine-tuned models in Figure 1. $\mathcal{H}(\widetilde{\Theta})$ is the pretrained network, where $\widetilde{\Theta}$ include parameters of the embedding and convolutional layers; $w(\hat{\Theta})$ and $\pi(\nu)$ represent the classification layers for pre-training and fine-tuning, respectively; and $\widetilde{\mathcal{H}}(\widetilde{\Theta}; \vartheta)$ is the fine-tuning network with pre-trained $\widetilde{\Theta}$ and re-learned model patch parameters ϑ. $\mathcal{H}(\widetilde{\Theta})$ and $\widetilde{\mathcal{H}}(\widetilde{\Theta}; \vartheta)$ share the same network architecture except the injected model patches (explained later).

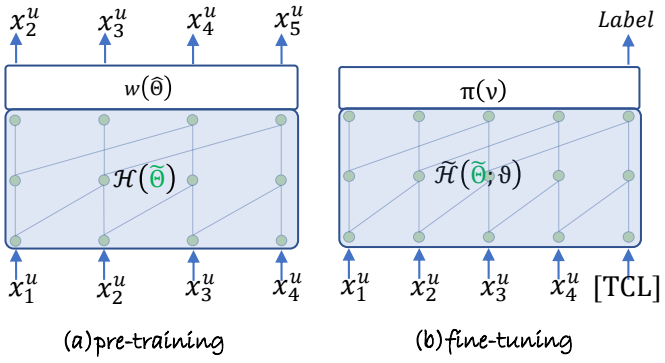


Figure 1: Illustration of parameters in PeterRec. x_i^udenote an itemID in the input sequence of user u. [TCL] is a special token representing the classification symbol.

3.2 User Representation Pre-training

Pre-training Objectives. Following NextItNet [37], we model the user interaction dependencies in the sequence by a left-to-right chain rule factorization, aka an autoregressive [2] method. Mathematically, the joint probability $p(\mathbf{x}^u;\Theta)$ of each user sequence is represented by the product of the conditional distributions over the items, as shown in Figure 1 (a):

$$p(\mathbf{x}^u;\Theta) = \prod_{i=1}^{n} p(x_i^u \,|x_1^u, .., x_{i-1}^u;\Theta) \tag{1}$$

where the value $p(x_i^u|x_1^u, ..., x_{i-1}^u;\Theta)$ is the probability of the i-th interacted item x_i^u conditioned on all its previous interactions $\{x_1^u, ..., x_{i-1}^u\}$, Θ is the parameters of pre-trained model including network parameters $\widetilde{\Theta}$ and the classification layer parameters $\widehat{\Theta}$. With such a formulation, the interaction dependencies in $\mathbf{x}^u$ can be explicitly modeled, which is more powerful than existing pre-training approaches (e.g., DUPN) that simply treat the item sequence $\mathbf{x}^u$ as common feature vectors. To the best of our knowledge, PeterRec is the first TL model in the recommender system domain that is pre-trained by unsupervised (or self-supervised) autoregressive approach.

Even though user-item interactions come in the form of sequence data, the sequential dependency may not be strictly held in terms of user preference, particularly for recommendations. This has been verified in [36], which introduced GRec that estimates the target interaction by considering both past and future interactions. As such, we introduce an alternative pre-training objective by taking account of two-side contexts. Specifically, we randomly mask a certain percentage of items (e.g., 30%) of $\mathbf{x}^u$ by filling in the mask symbols (e.g., "__") in the sequence, and then predict the items at these masked position by directly adding a softmax layer on the encoder of GRec.

Formally, let $\mathbf{x}_{\triangle}^u = \{x_{\triangle_1}^u, ..., x_{\triangle_m}^u\}$ $(1 \leq m < t)$ be the masked interactions, and $\tilde{\mathbf{x}}^u$ is the sequence of $\mathbf{x}^u$ by replacing items in $\mathbf{x}_{\triangle}^u$ with "__", the probability of $p(\mathbf{x}_{\triangle}^u)$ is given as:

$$p(\mathbf{x}_{\triangle}^u;\Theta) = \prod_{i=1}^{m} p(x_{\triangle_i}^u \,|\tilde{\mathbf{x}}^u;\Theta) \tag{2}$$

To maximize $p(\mathbf{x}^u;\Theta)$ or $p(\mathbf{x}_{\triangle}^u;\Theta)$, it is equivalent to minimize the cross-entropy (CE) loss $L(\mathcal{S};\Theta) = -\sum_{(u,\mathbf{x}^u)\in\mathcal{S}} \log p(\mathbf{x}^u;\Theta)$ and $G(\mathcal{S};\Theta) = -\sum_{(u,\mathbf{x}^u)\in\mathcal{S}} \log p(\mathbf{x}_{\triangle}^u;\Theta)$, respectively. It is worth mentioning that while similar pre-training objectives have been applied in the NLP [6] and computer vision [30] domains recently, the effectiveness of them remains completely unknown in recommender systems. Hence, in this paper instead of proposing a new pre-training objective function, we are primarily interested in showing readers what types of item recommendation models can be applied to user representation learning, and how to adapt them for pre-training & fine-tuning so as to bridge the gap between different domains.

Petrained Network Architectures. The main architecture ingredients of the pre-trained model are a stack of dilated convolutional (DC) [36, 37] layers with exponentially increased dilations and a repeatable pattern, e.g., {1, 2, 4, 8, 16, 32, 1, 2, 4, 8, 16, 32, ..., 32}. Every two DC layers are connected by a shortcut connection, called residual block [11]. Each DC layer in the block is followed by a layer normalization and non-linear layer activation, as illustrated in Figure 3 (a). As illustrated in [37] and [36], the pre-trained network should be built by causal and non-causal CNNs for objective fuctions of Eq. (1) and Eq. (2), respectively.

Concretely, the residual block with the DC operations is formalized as follows:

$$\mathcal{H}_{DC}(\boldsymbol{E}) = \begin{cases} \boldsymbol{E} + \mathcal{F}_{cauCNN}(\boldsymbol{E}) & optimized\ by \quad Eq.\ (1) \\ \boldsymbol{E} + \mathcal{F}_{non\_cauCNN}(\boldsymbol{E}) & optimized\ by \quad Eq.\ (2) \end{cases} \tag{3}$$

where $\boldsymbol{E} \in \mathbb{R}^{n\times k}$ and $\mathcal{H}_{DC}(\boldsymbol{E}) \in \mathbb{R}^{n\times k}$ are the input and output matrices of layers considered, k is the embedding dimension, $\boldsymbol{E}+\mathcal{F}$ is a shortcut connection by element-wise addition, and $\mathcal{F}_{cauCNN}(\boldsymbol{E})$ & $\mathcal{F}_{non\_cauCNN}(\boldsymbol{E})$ are the residual mappings as follows

$$\begin{aligned} \mathcal{F}_{cauCNN}(\boldsymbol{E}) &= \sigma(\mathcal{LN}_2(\psi_2(\sigma(\mathcal{LN}_1(\psi_1(\boldsymbol{E})))))) \\ \mathcal{F}_{non\_cauCNN}(\boldsymbol{E}) &= \sigma(\mathcal{LN}_2(\phi_2(\sigma(\mathcal{LN}_1(\phi_1(\boldsymbol{E})))))) \end{aligned} \tag{4}$$

where ψ and ϕ represent causal (e.g., Figure 2 (a) & (b) and non-causal (e.g., (c) & (d)) convolution operations, respectively, and the biases are omitted for shortening notations. $\mathcal{LN}$ and σ represent layer normalization [1] and ReLU [23], respectively.

3.3 Adapting Representations

After the above pre-training process, we can adapt the learned representations to specific downstream tasks. The primary goal here is to develop a fine-tuning framework that works well in multiple-domain settings by introducing only a small fraction of domain-specific parameters, and attain a high-degree of parameter sharing between domains. Specifically, the architecture of fine-tuning PeterRec contains three components as shown in Figure 1 (b): all except the classification layer of the pre-trained model (parameterized by $\widetilde{\Theta}$), the new classification layer (parameterized by ν) for the corresponding downstream task, and the model patches (parameterized by ϑ) that are inserted in the pre-trained residual blocks. In the following, we first present the overall fine-tuning framework. Then, we describe the details of the model patch structure and show how to inject it into the pre-trained model.

Fine-tuning Framework. Let assume that the model patches have been inserted and initialized in the pre-trained model. The overall architectures of PeterRec are illustrated in Figure 2. As a running example, we describe in detail the fine-tuning procedures using the causal CNN network, as shown in (a). For each instance

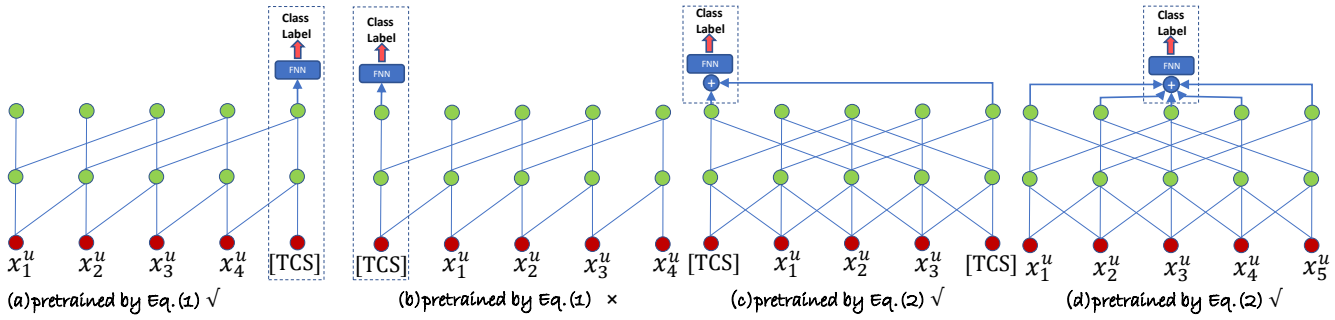


Figure 2: The fine-tuning architecture of PeterRec illustrated with one residual block. Each layer of (green) neurons corresponds to a DC layer in Figure 3. The normalization layer, ReLU layers and model patches are not depicted here for clearity. (a)(b) and (c)(d) are causal and non-causal convolutions, respectively. FNN is the feedforward neural network for classification with parameter ν. (a)(c) and (d) are suggested fine-tuning architectures. (b) is not correct since no information can be obtained by causal convolution if [TCS] is inserted at the beginning.

(u, y) in $\mathcal{T}$, we first add a [TCL] token at the end position of user sequence u, and achieve the new input, i.e., $\mathbf{x}^u = \{x_1^u, ..., x_n^u, [\text{TCL}]\}$. Then, we feed this input sequence to the fine-tuning neural network. By performing a series of causal CNN operations on the embedding of $\mathbf{x}^u$, we obtain the last hidden layer matrix. Afterwards, a linear classification layer is placed on top of the final hidden vector of the [TCL] token, denoted by $\boldsymbol{h}_n \in \mathbb{R}^k$. Finally, we are able to achieve the scores $\boldsymbol{o} \in \mathbb{R}^{|\mathcal{Y}|}$ with respect to all labels in $\mathcal{Y}$, and the probability to predict y.

$$\begin{aligned} \boldsymbol{o} &= \boldsymbol{h}_n \boldsymbol{W} + \boldsymbol{b} \\ p(y|u) &= p(y|\mathbf{x}^u) = softmax(\boldsymbol{o}_y) \end{aligned} \tag{5}$$

where $\boldsymbol{W} \in \mathbb{R}^{k \times |\mathcal{Y}|}$ and $\boldsymbol{b} \in \mathbb{R}^{|\mathcal{Y}|}$ are the projection matrix and bias term.

In terms of the pre-trained model by non-causal CNNs, PeterRec can simply add [TCS]s at the start and the end positions of $\mathbf{x}^u$, as shown in Figure 2 (c), i.e., $\mathbf{x}^u = \{[\text{TCL}], x_1^u, ..., x_n^u, [\text{TCL}]\}$, and accordingly[1]

$$\boldsymbol{o} = (\boldsymbol{h}_0 + \boldsymbol{h}_n)\boldsymbol{W} + \boldsymbol{b} \tag{6}$$

Alternatively, PeterRec can use the sum of all hidden vectors of $\boldsymbol{h}$ without adding any [TCL] for both causal and non-causal CNNs, e.g., Figure 2 (d).

$$\boldsymbol{o} = (\sum_{i=1}^{n} \boldsymbol{h}_i)\boldsymbol{W} + \boldsymbol{b} \tag{7}$$

Throughout this paper, we will use Figure 2 (a) for causal CNN and (c) for non-causal CNN in our experiments.

As for the fine-tuning objective functions of PeterRec, we adopt the pairwise ranking loss (BPR) [28] for top-N item recommendation task and the CE loss for the user profile classification tasks.

$$\begin{aligned} R_{BPR}(\mathcal{T}; \tilde{\Theta}; \nu; \vartheta) &= -\sum_{(u,y) \in \mathcal{T}} \log \delta(\boldsymbol{o}_y - \boldsymbol{o}_{y^-}) \\ R_{CE}(\mathcal{T}; \tilde{\Theta}; \nu; \vartheta) &= -\sum_{(u,y) \in \mathcal{T}} \log p(y|u) \end{aligned} \tag{8}$$

where δ is the logistic sigmoid function, and y^- is a false label randomly sampled from $\mathcal{Y} \backslash y$ following [28]. Note that in [35, 38], authors showed that a properly developed dynamic negative sampler usually performed better than the random one if $|\mathcal{Y}|$ is huge. However, this is beyond the scope of this paper, and we leave it as future investigation.

Eq.(8) can be then optimized by SGD or its variants such as Adam [19]. For each downstream task, PeterRec only updates ϑ and ν (including $\boldsymbol{W}$ & $\boldsymbol{b}$) by freezing pre-trained parameters $\widetilde{\Theta}$. The update equation is given as

$$\begin{aligned} \vartheta_{new} &= \vartheta_{old} - \eta \frac{\partial R(\widetilde{\Theta}; \nu; \vartheta)}{\partial \vartheta} \\ \nu_{new} &= \nu_{old} - \eta \frac{\partial R(\widetilde{\Theta}; \nu; \vartheta)}{\partial \nu} \end{aligned} \tag{9}$$

where η is the learning rate. We omit the descriptions of gradient equations $\frac{\partial R(\tilde{\Theta}; \nu; \vartheta)}{\partial \vartheta}$ & $\frac{\partial R(\tilde{\Theta}; \nu; \vartheta)}{\partial \nu}$, which can be automatically calculated by chain rule in deep learning libraries, such as Tensorflow[2].

Model Patch Structure. The model patch is an additional parametric neural network, which adapts the pre-trained DC residual blocks to corresponding tasks. Our work is motivated and inspired by recent learning-to-learn approaches in [5] which show that it is possible to predict up to 95% of the model parameters given only the remaining 5%. Rather than predicting parameters, we aim to demonstrate how to modify the values of the pre-trained model to obtain better accuracy in related but very different tasks by training only few parameters.

The structure of the model patch neural network is shown in Figure 3 (f). We construct it using a simple residual network (ResNet) architecture with two 1×1 convolutional layers considering its strong learning capacity in the literature [11]. To minimize the number of parameters, we propose a bottlenet architecture. Specifically, the model patch consists of a projection-down layer, an activation function, a projection-up layer, and a shortcut connection, where the projection-down layer projects the original k dimensional channels to d ($d \ll k$, e.g., $k = 8d$)[3] by $1 \times 1 \times k \times d$ convolutional operations ϕ_{down}, and the projection-up layer is to project it back to its original dimension by $1 \times 1 \times d \times k$ convolutional operations ϕ_{up}. Formally, given its input tensor $\widetilde{\boldsymbol{E}}$, the output of the model patch can be expressed as:

$$\mathcal{H}_{MP}(\widetilde{\boldsymbol{E}}) = \widetilde{\boldsymbol{E}} + \phi_{up}(\sigma(\phi_{down}(\widetilde{\boldsymbol{E}}))) \tag{10}$$

[1] [TCS] and the blank token, i.e., "_", used in the pre-training phase can be the represented by a same index value.

[2] https://www.tensorflow.org

[3] Throughout this paper, we used $k = 8d$, though we did not find that $k = 16d$ degraded accuracy.

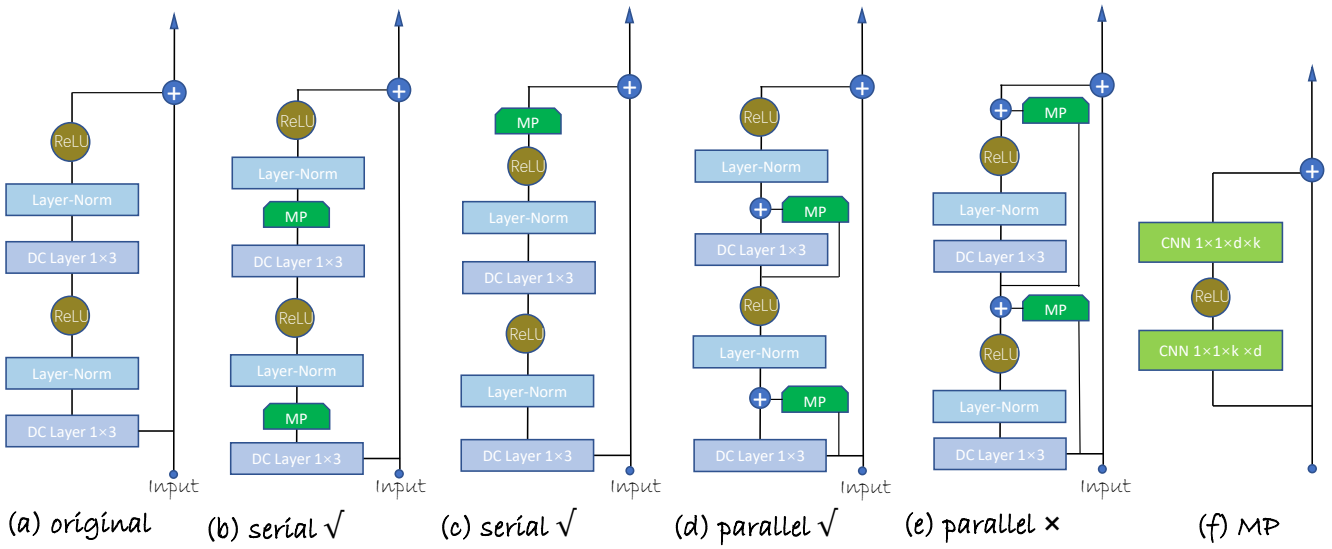


Figure 3: Model patch (MP) and insertion methods. (a) is the original pre-trained residual block; (b) (c) (d) (e) are the fine-tuned residual blocks with inserted MPs; and (f) is the MP block. + is the addition operation. 1×3 is the kernel size of dilated convolutional layer.

Suppose that the kernel size of the original dilated convolutions is 1×3, the total number of the parameters of each DC layer is $3 * k^2 = 192d^2$, while the number of the patched neural network is $2 * k * f = 16d^2$, which is less than 10% parameters of the original DC network. Note that parameters of biases and layer normalization are not taken into account since the numbers are much smaller than that of the DC layer. Note that using other similar structures to construct the model patch may also perform well, such as in [33], but it generally needs to meet three requirements: (1) to have a much smaller scale compared with the original convolutional neural network; (2) to guarantee that the pre-trained parameters are left untouched during fine-tuning; and (3) to attain good accuracy.

Insertion Methods. Having developed the model patch architecture, the next question is how to inject it into the current DC block. We introduce two ways for insertion, namely serial & parallel mode patches, as shown in Figure 3 (b) (c) (d) and (e).

First, we give the formal mathematical formulations of the fine-tuning block (by using causal CNNs as an example) as follows:

$$\widetilde{\mathcal{H}}_{DC}(\boldsymbol{E}) = \boldsymbol{E} + \widetilde{\mathcal{F}}_{cauCNN}(\boldsymbol{E}) \tag{11}$$

where $\widetilde{\mathcal{F}}_{cauCNN}$, short for $\widetilde{\mathcal{F}}$ below, is

$$\widetilde{\mathcal{F}} = \begin{cases} \sigma(\mathcal{LN}_2(\mathcal{H}_{MP2}(\psi_2(\sigma(\mathcal{LN}_1(\mathcal{H}_{MP1}(\psi_1(\boldsymbol{E})))))))) & Figure\ 3\ (b) \\ \mathcal{H}_{MP}(\sigma(\mathcal{LN}_2(\psi_2(\sigma(\mathcal{LN}_1(\psi_1(\boldsymbol{E})))))))) & Figure\ 3\ (c) \\ \sigma(\mathcal{LN}_2(\mathcal{H}_{MP2}(\widetilde{\boldsymbol{E}}) + \psi_2(\widetilde{\boldsymbol{E}})), & Figure\ 3\ (d) \\ \quad where\ \widetilde{\boldsymbol{E}} = \sigma(\mathcal{LN}_1(\mathcal{H}_{MP1}(\boldsymbol{E}) + \psi_1(\boldsymbol{E})))) & \\ \mathcal{H}_{MP2}(\widetilde{\boldsymbol{E}}) + \sigma(\mathcal{LN}_2(\psi_2(\widetilde{\boldsymbol{E}}))), & Figure\ 3\ (e) \\ \quad where\ \widetilde{\boldsymbol{E}} = \mathcal{H}_{MP1}(\boldsymbol{E}) + \sigma(\mathcal{LN}_1(\psi_1(\boldsymbol{E}))) & \end{cases} \tag{12}$$

In fact, as shown in Figure 3, we only suggest architectures of (b) (c) and (d) as (e) usually converges and performs significantly worse as evidenced and explained in Section 4.5. Here, we give several empirical principles on how to insert this model.

- For the serial insertion, the inserted positions are very flexible so that one can inject the patch neural network either before or after layer normalization, as shown in (b) and (c).
- For the serial insertion, the number of model patches for each DC residual block is very flexible so that one can inject either one or two patches. It gives almost the same results if k in (c) is two times larger than that in (b).
- For parallel insertion, PeterRec is sensitive to the inserted positions, as shown in (d) and (e). Specifically, the model patch that is injected before layer normalization (i.e., (d)) performs better than that between layer normalization and activation function, which performs largely better than that after activation function (i.e., (e)) .
- For parallel insertion, PeterRec with two model patches inserted in the DC block usually performs slightly better than that with only one patch.

In practice, both the serial and parallel insertions with a proper design can yield comparable results with fine-tuning the entire model. Let us give a quantitative analysis regarding the number of tuned parameters. Assuming that PeterRec utilizes 500,000 items from a source domain, 1024 embedding & hidden dimensions, 20 residual blocks (i.e., 40 layers), and 1000 class labels to be predicted in the target domain, the overall parameters are $500,000 * 1024 + 1024 * 1024 * 3$ (here 3 is the kernel size) $* 40 + 1024 * 1000 \approx 639$ million, the number of tuned parameters for ϑ and ν is $2 * 1024 * 1024/8 * 40 \approx 10$ million and $1024 * 1000 \approx 1$ million, respectively, which in total takes less than 1.7% of the number of all parameters. Note that (1) the number of parameters ν can never be shared due to the difference of the output space in the target task, and it depends on the specific downstream task. It may be large if the task is an item recommendation task and may be very small if the task is user modeling (E.g., for gender estimation, it is $1024 * 2 = 2048$); (2) Though there are several ways to compress the input

Table 1: Number of instances. Each instance in $\mathcal{S}$ and $\mathcal{T}$ represents $(u, \mathbf{x}^u)$ and (u, y) pairs, respectively. The number of source items $|\mathcal{X}|$=191K, 645K, 645K, 645K, 645K (K = 1000), and the number of target labels $|\mathcal{Y}|$=20K,17K, 2, 8, 6 for the five dataset from left to right in the below table.

Domain	ColdRec-1	ColdRec-2	GenEst	AgeEst	LifeEst
$\mathcal{S}$	1,649,095	1,551,881	-	-	-
$\mathcal{T}$	3,798,114	2,947,688	1,548,844	1,551,357	1,075,010

embedding and output classification layers, which can lead to really large compression rates [21], we do not describe them in detail as this is clearly beyond the scope of our paper.

4 EXPERIMENTS

In our experiments, we answer the following research questions:

(1) **RQ1:** Is the self-supervised learned user representation really helpful for the downstream tasks? To our best knowledge, as a fundamental research question for transfer learning in the recommender system domain, this has never been verified before.
(2) **RQ2:** How does PeterRec perform with the proposed model patch compared with fine-tuning the last layer and fine-tuning the entire model?
(3) **RQ3:** What types of user profiles can be estimated by PeterRec? Does PeterRec work well when users are cold or new in the target domain.
(4) **RQ4:** Are there any other interesting insights we can draw by the ablation analysis of PeterRec? E.g., which type of pre-trained network helps to yield better fine-tuning accuracy? Does a pre-trained model that performs better on the source task also perform better on the target tasks?

4.1 Experimental Settings

Datasets. We conduct experiments on several large-scale industrial datasets collected by the Platform and Content Group of Tencent[4].

1. ColdRec-1 dataset: This contains both source and target datasets. The source dataset is the news recommendation data collected from QQ Browser[5] recommender system from 19th to 21st, June, 2019. Each interaction denotes a positive feedback by a user at certain time. For each user, we construct the sequence using his recent 50 watching interactions by the chronological order. For users that have less than 50 interactions, we simply pad with zero in the beginning of the sequence following common practice [37]. The target dataset is collected from Kandian[6] recommender system in the same month where an interaction can be be a piece of news, a video or an advertisement. All users in Kandian are cold with at most 3 interactions (i.e., $g \leq 3$) and half of them have only one interaction. All users in the target dataset have corresponding records in the source dataset.

2. ColdRec-2 dataset: It has similar characteristics with ColdRec-1. The source dataset contains recent 100 watching interactions of each user, including both news and videos. The users in the target dataset have at most 5 interactions (i.e., $g \leq 5$), and half of them have only one interaction.

3. GenEst dataset: It has only a target dataset since all users are from the source dataset of ColdRec-2. Each instance in GenEst is a user and his gender (male or female) label ($g = 1$) obtained by the registration information.

4. AgeEst dataset: Similar to GenEst, each instance in AgeEst is a user and his age bracket label ($g = 1$) — one class represents 10 years.

5. LifeEst dataset: Similar to GenEst, each instance in LifeEst is a user and his life status label ($g = 1$), e.g., single, married, pregnancy or parenting.

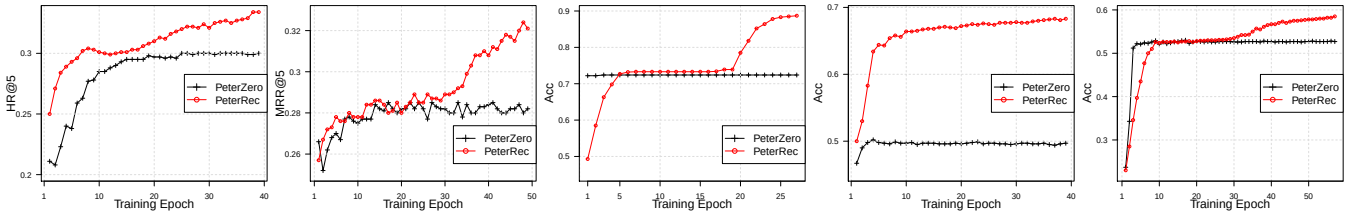
Table 1 summarizes other statistics of evaluated datasets.

Evaluation Protocols. To evaluate the performance of PeterRec in the downstream tasks, we randomly split the target dataset into training (70%), validation (3%) and testing (27%) sets. We use two popular top-5 metrics — MRR@5 (Mean Reciprocal Rank) [38] and HR@5 (Hit Ratio) [13] — for the cold item recommendation datasets (i.e. ColdRecs), and the classification accuracy (denoted by Acc, where Acc = number of correct predictions/total number of predictions) for the other three datasets. Note that to speed up the experiments of item recommendation tasks, we follow the common strategy in [13] by randomly sampling 99 negative examples for the true example, and evaluate top-5 accuracy among the 100 items.

Compared Methods. We compare PeterRec with the following baselines to answer the proposed research questions.

- To answer **RQ1**, we compare PeterRec in two cases: well-pre-trained and no-pre-trained settings. We refer to PeterRec with randomly initialized weights (i.e., without pre-training) as PeterZero.
- To answer **RQ2**, we compare PeterRec with three baselines which initialize their weights using pre-trained parameters: (1) fine-tuning only the linear classification layer that is designed for the target task, i.e., treating the pre-trained model as a feature extractor, referred to as FineCLS; (2) fine-tuning the last CNN layer, along with the linear classification layer of the target task, referred to as FineLast; (3) fine-tuning all parameters, including both the pre-trained component (again, excluding its softmax layer) and the linear classication layer for the new task, referred to as FineAll.
- To answer **RQ3**, we compare PeterRec with an intuitive baseline, which performs classification based on the largest number of labels in $\mathcal{T}$, referred to as LabelCS. For cold-start user recommendation, we compare it with two powerful baseline NeuFM [12] and DeepFM [9]. The interacted items in $\mathcal{S}$ are treated as features for the two models, which can be seen as a feature-based transfer learning. In adition, we also present a multi-task learning (referred to as MTL) baseline by adapting DUPN [24] to our dataset, which jointly learns the objective functions of both source and target domains instead of using the two-stage pre-training and fine-tuning schemes of PeterRec. Note that PeterRec is not directly comparable with DUPN since (1) its training process relies on the a large number of item and user behavior (e.g., click, bookmark and purchase, etc.) features; (2) for each user, it requires his

[4] https://www.tencent.com/en-us/
[5] https://browser.qq.com
[6] https://sdi.3g.qq.com/v/2019111020060911550



(a) Cold-Rec1 (one epoch: $50 * b$) **(b)** Cold-Rec2 (one epoch: $50 * b$) **(c)** GenEst (one epoch : $5 * b$) **(d)** AgeEst (one epoch: $50 * b$) **(e)** LifeEst (one epoch: $5 * b$)

Figure 4: Impact of pre-training — PeterRec (not converged) vs. PeterZero (fully converged) with the causal CNN. b is batch size. Note that since PeterZero converges much faster (and worse) in the first several epoches, here we only show the results of PeterRec for these beginning epoches for a better comparison. The converged results are given in Table 2.

Table 2: Impacts of pre-training — FineZero vs. FineAll (with the causal CNN architectures). Without special mention, in the following we only report ColdRec-1 with HR@5 and ColdRec-2 with MRR@5 for demonstration.

Model	ColdRec-1	ColdRec-2	GenEst	AgeEst	LifeEst
FineZero	0.304	0.290	0.900	0.703	0.596
FineAll	0.349	0.333	0.903	0.710	0.610
PeterRec	0.348	0.334	0.903	0.708	0.610

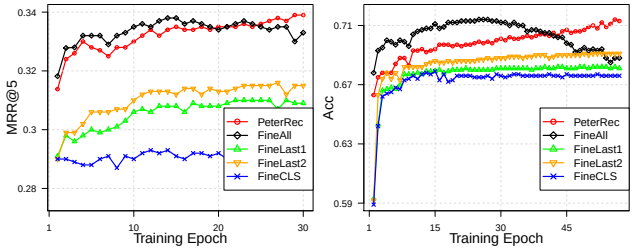
Table 3: Performance comparison (with the non-causal CNN architectures). The number of fine-tuned parameters (ϑ and ν) of PeterRec accounts for 9.4%, 2.7%, 0.16%, 0.16%, 0.16% of FineAll on the five datasets from left to right.

Model	ColdRec-1	ColdRec-2	GenEst	AgeEst	LifeEst
FineCLS	0.295	0.293	0.900	0.679	0.606
FineLast	0.330	0.310	0.902	0.682	0.608
FineAll	0.352	0.338	0.905	0.714	0.615
PeterRec	0.351	0.339	0.906	0.714	0.615

other preference labels (e.g., conversion rate, price or fashion styles) to perform multi-task learning.

- To answer **RQ4**, we compare PeterRec by using different settings, e.g., using causal and non-causal CNNs as explained in Section 3.2, referred to as PeterRecal and PeterRecon, respectively.

Hyper-parameter Details. All models were trained on GPUs (Tesla P40) using Tensorflow. All reported results use an embedding & hidden dimension of k=256. Results for k=128, 512 show similar conclusions but are omitted for saving space. The learning rates for Adam [19] with η = 0.01, 0.001 and 0.0001 give similar behaviors. As a compromise of speed and accuracy, we use η = 0.001 for all compared models on the first two datasets and η = 0.0001 on the other three datasets. All models including causal and non-causal CNNs use dilation $\{1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4, 8\}$ (16 layers or 8 residual blocks in total) following NextItNet [37]. Further increasing CNN layers or k does not yield significantly better results for these datasets. Batch size b and kernel size are set to 512 and 3 respectively for all models.



(a) Cold-Rec2 (one epoch: $500 * b$) **(b)** AgeEst (one epoch: $500 * b$)

Figure 5: Convergence behaviors of PeterRec and baselines (with the non-causal CNN). FineLast1 and FineLast2 denote FineLasts that optimize only the last one and two CNN layers (including the corresponding layer normalizations), respectively. All models here have fully converged. The number of parameters to be re-learned: FineAll≫ FineLast2> PeterRec≈FineLast1>FineCLS.

As for the pre-trained model, we use 90% of the dataset in $\mathcal{S}$ for training, and the remaining for validation. Different from fine-tuning, the measures for pre-training (i.e., MRR@5) are calculated based on the rank in the whole item pool following [37]. We use η = 0.001 for all pre-trained models. Batch size is set to 32 and 128 for causal and non-causal CNNs due to the consideration of memory space. The masked percentage for non-causal CNNs is 30%. Other parameters, such as kernel size and dilations are kept the same as mentioned above.

4.2 RQ1.

Since PeterRec has a variety of variants with different circumstances (e.g., causal and non-causal versions, different insert methods (see Figure 3), and different fine-tuning architectures (see Figure 2)), presenting all results on all the five datasets is redundant and space unacceptable. Hence, in what follows, we report parts of the results with respect to some variants of PeterRec (on some datasets or metrics) considering that their behaviors are consistent.

To answer RQ1, we report the results in Figure 4 & Table 2. For all compared models, we use the causal CNN architecture. For PeterRec, we use the serial insertion in Figure 3 (c). First, we observe that PeterRec outperforms PeterZero with large improvements on all the five datasets. Since PeterRec and PeterZero use exactly the same network architectures and hyper-parameters, we can draw

Table 4: Results regarding user profile prediction. Interestingly, we find that the number of male users is much larger than females with the ratio of 72.5% : 27.5% in the ground truth of GenEst.

Model	GenEst	AgeEst	LifeEst
LabelCS	0.725	0.392	0.446
MTL	0.899	0.706	0.599
PeterRec	0.906	0.714	0.615

Table 5: Top-5 Accuracy in the cold user scenario.

Data	NeuFM	DeepFM	MTL	PeterRec
ColdRec-1	0.335	0.326	0.337	0.351
ColdRec-2	0.321	0.317	0.319	0.339

the conclusion that the self-supervised pre-trained user representation is of great importance in helping improve the accuracy of downstream tasks. To further verify it, we also report results of FineAll and FineZero in Table 2. Similarly, FineAll largely exceeds FineZero (i.e., FineAll with random initialization) on all datasets. The same conclusion also applies to FineCLS and FineLast with their random initialization variants.

4.3 RQ2.

To answer RQ2, we report the results in Table 3. We use the non-causal CNN architecture for all models and parallel insertion for PeterRec. First, we observe that with the same pre-training model, FineCLS and FineLast perform much worse than FineAll, which demonstrates that fine-tuning the entire model benefits more than tuning only the last (few) layers. Second, we observe that PeterRec achieves similar results with FineAll, which suggests that fine-tuning the proposed model patch (MP) is as effective as fine-tuning the entire model. By contrast, PeterRec retains most pre-trained parameters (i.e., $\tilde{\Theta}$) unchanged for any downstream task, whereas FineAll requires a large separate set of parameters to be re-trained and saved for each task, and thus is not efficient for resource-limited applications and multi-domain learning settings. Moreover, fine-tuning all parameters may easily cause the overfitting (see Figure 5 (b) and Figure 6) problems. To clearly see the convergence behaviors of these models, we also plot their results on the ColdRec and AgeEst datasets in Figure 5.

4.4 RQ3.

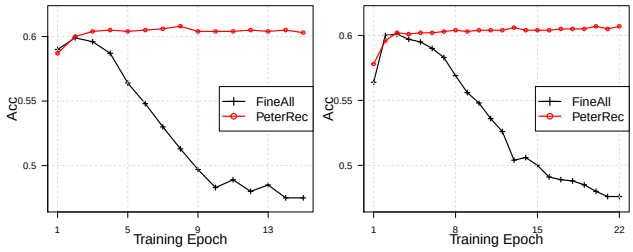
To answer RQ3, we demonstrate the results in Table 4 and 5. Clearly, PeterRec notably outperforms LabelCS, which demonstrates its the effectiveness in estimating user profiles. Meanwhile, PeterRec yields better top-5 accuracy than NeuFM, DeepFM and MTL in the cold-user item recommendation task. Particularly, PeterRec outperforms MTL in all tasks, which implies that the proposed two-stage pre-training & fine-tuning paradigm is more powerful than the joint training in MTL. We argue this is because the optimal parameters learned for two objectives in MTL does not gurantee optimal performance for fine-tuning. Again, compared with these baselines, PeterRec is memory-efficient since it only maintains a small set of model patch parameters for a new task while others have to store all parameters for each task. In addition, the training speed of MTL is several times slower than PeterRec due to the expensive pre-training objective functions. If there are a large number of sub-tasks, PeterRec will always be a better choice considering its high degree of parameter sharing. To the best of our knowledge, PeterRec is the first model that considers the memory-efficiency issue for multi-domain recommendations.

Table 6: PeterRecal vs. PeterRecon. The results of the first and last two columns are ColdRec-1 and AgeEst datasets, respectively.

Model	PeterRecal	PeterRecon	PeterRecal	PeterRecon
Pretaining	0.023	0.020	0.045	0.043
Fine-tuning	0.348	0.351	0.708	0.714

Table 7: Performance of different insertions in Figure 3 on AgeEst.

Model	(b)	(c)	(d)	(e)
PeterRecal	0.708	0.708	0.708	0.675
PeterRecon	0.710	0.710	0.714	0.685



(a) 5 % of training data (one epoch: $500 * b$)**(b)** 10 % of training data (one epoch: $500 * b$)

Figure 6: Convergence behaviors of PeterRec and FineAll on LifeEst using much less training data. The improvements of PeterRec relative to FineAll is around 1.5% and 1.7% on (a) and (b) respectively in terms of the optimal performance.

4.5 RQ4.

This subsection offers several insightful findings: (1) By contrasting PeterRecal and PeterRecon in Table 6, we can draw the conclusion that better pre-training models for sequential recommendation may not necessarily leads to better transfer-learning accuracy. This is probably because PeterRecon takes two-side contexts into consideration [36], which is more effective than the sequential patterns learned by PeterRecal for these downstream tasks. However, for the same model, better pre-training models usually lead to better fine-tuning performance. Such results are simply omitted due to limited space. (2) By comparing results in Table 7, we observe that for parallel insertion, the MP has to be inserted before the normalization layer. We argue that the parallelly inserted MP in Figure 3 (e) may break up the addition operation in the original residual block architecture (see FIgure 3 (a)) since MP in (e) introduces two additional summation operations, including the sum in MP and

sum with the ReLU layer. (3) In practice, it is usually very expensive to collect a large amount of user profile data, hence we present the results with limited training examples in Figure 6. As clearly shown, with limited training data, PeterRec performs better than FineAll, and more importantly, PeterRec is very stable during fine-tuning since only a fraction of parameters are learned. By contrast, FineAll has a severe overfitting issue, which cannot be solved by regularization or dropout techniques.

5 CONCLUSIONS

In this paper, we have shown that (1) it is possible to learn universal user representations by modeling only unsupervised user sequential behaviors; and (2) it is also possible to adapt the learned representations for a variety of downstream tasks. By introducing the model patch, PeterRec allows all pre-trained parameters unchanged during fine-tuning, enabling efficent & effective adaption to multiple domains with only a small set of re-learned parameters for a new task. We have evaluated several alternative designs of PeterRec, and made insightful observations by extensive ablation studies. By releasing both high-quality datasets and codes, we hope PeterRec serves as a benchmark for transfer learning in the recommender system domain.

We believe PeteRec can be applied in more domains aside from tasks in this paper. For example, if we have the video watch behaviors of a teenager, we may know whether he has depression or propensity for violence by PeterRec without resorting to much feature engineering and human-labeled data. This can remind parents taking measures in advance to keep their children free from such issues. For future work, we may explore PeteRec with more tasks.

REFERENCES

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

[2] Yoshua Bengio and Samy Bengio. 2000. Modeling high-dimensional discrete data with multi-layer neural networks. In *Advances in Neural Information Processing Systems*. 400–406.

[3] Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. 2016. Learning feed-forward one-shot learners. In *Advances in Neural Information Processing Systems*. 523–531.

[4] Chong Chen, Min Zhang, Chenyang Wang, Weizhi Ma, Minming Li, Yiqun Liu, and Shaoping Ma. 2019. An Efficient Adaptive Transfer Neural Network for Social-aware Recommendation. (2019).

[5] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. 2013. Predicting parameters in deep learning. In *Advances in neural information processing systems*. 2148–2156.

[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] Samuel Dodge and Lina Karam. 2017. A study and comparison of human and deep learning recognition performance under visual distortions. In *2017 26th international conference on computer communication and networks (ICCCN)*. IEEE, 1–7.

[8] Guibing Guo, Shichang Ouyang, Xiaodong He, Fajie Yuan, and Xiaohua Liu. 2019. Dynamic item block and prediction enhancing block for sequential recommendation. International Joint Conferences on Artificial Intelligence Organization.

[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] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. 2019. SpotTune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4805–4814.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.

[12] 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.

[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 173–182.

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

[15] 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. *arXiv preprint arXiv:1902.00751* (2019).

[16] Guangneng Hu, Yu Zhang, and Qiang Yang. 2018. Conet: Collaborative cross networks for cross-domain recommendation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 667–676.

[17] Guangneng Hu, Yu Zhang, and Qiang Yang. 2018. MTNet: a neural approach for cross-domain recommendation with unstructured text.

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

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

[20] Simon Kornblith, Jonathon Shlens, and Quoc V Le. 2019. Do better imagenet models transfer better?. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2661–2671.

[21] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[22] Pramod Kaushik Mudrakarta, Mark Sandler, Andrey Zhmoginov, and Andrew Howard. 2018. K For The Price Of 1: Parameter Efficient Multi-task And Transfer Learning. *arXiv preprint arXiv:1810.10703* (2018).

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

[24] 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.

[25] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. [n. d.]. Improving language understanding by generative pre-training. ([n. d.]).

[26] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*. 506–516.

[27] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2018. Efficient parametrization of multi-domain deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8119–8127.

[28] 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*. AUAI Press, 452–461.

[29] Asa Cooper Stickland and Iain Murray. 2019. BERT and PALs: Projected Attention Layers for Efficient Adaptation in Multi-Task Learning. *arXiv preprint arXiv:1902.02671* (2019).

[30] Weijie Su, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. 2019. Vl-bert: Pre-training of generic visual-linguistic representations. *arXiv preprint arXiv:1908.08530* (2019).

[31] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *ACM International Conference on Web Search and Data Mining*.

[32] 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*. ACM, 1703–1712.

[33] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1492–1500.

[34] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in neural information processing systems*. 3320–3328.

[35] 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.

[36] Fajie Yuan, Xiangnan He, Haochuan Jiang, Guibing Guo, Jian Xiong, Zhezhao Xu, and Yilin Xiong. 2019. Future Data Helps Training: Modeling Future Contexts for Session-based Recommendation. *The world wide web conference* (2019).

[37] 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.

[38] Fajie Yuan, Xin Xin, Xiangnan He, Guibing Guo, Weinan Zhang, Chua Tat-Seng, and Joemon M Jose. 2018. fBGD: Learning embeddings from positive unlabeled data with BGD. (2018).

[39] 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).

[40] Kui Zhao, Yuechuan Li, Zhaoqian Shuai, and Cheng Yang. 2018. Learning and Transferring IDs Representation in E-commerce. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1031–1039.