LEARNING TWO-TIME-SCALE REPRESENTATIONS FOR LARGE SCALE RECOMMENDATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a surprisingly simple but effective two-time-scale (2TS) model for learning user representations for recommendation. In our approach, we will partition users into two sets, active users with many observed interactions and inactive or new users with few observed interactions, and we will use two RNNs to model them separately. Furthermore, we also design a two-stage training method for our model, where, in the first stage, we learn transductive embeddings for users and items, and then, in the second stage, we learn the two RNNs leveraging the transductive embeddings trained in the first stage. Our 2TS model allows us to achieve a nice bias-variance trade-off while being computationally efficient. In large scale datasets, our 2TS model is able to achieve significantly better recommendations than previous state-of-the-art, yet being much more computationally efficient.

1 INTRODUCTION

A hypothetical user's interaction with recommendation systems gives us *diminishing* return in terms of its information value in understanding the user. For an *active* user who already has lots of historical interactions with the system, she is typically well understood by the recommender, and each new interaction gives relatively little new information about this user. In contrast, for an *inactive* or *new* user, every additional interaction will provide interesting information for understanding this user. Therefore, the representations for active and inactive users should be updated differently when a new interaction occurs. Figure 1 illustrates such information diminishing phenomenon, where the amount of change in user embedding from ϕ^t to ϕ^{t+1} due to an additional interaction is decaying. One can select a particular threshold t^* for the number of interactions, above which the users can be categorized to active users, and below which inactive users. Roughly active users' embeddings evolve slowly as a function of the number of interactions, while inactive users' embeddings evolve fast as new interactions arrive. Hence a two-time-scale embedding evolution.

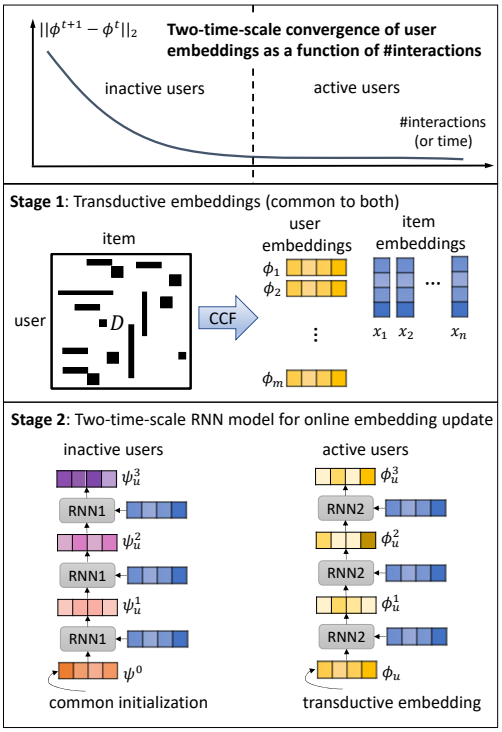


Figure 1: Two-time-scale convergence of user embeddings motivates us to design a two-stage method, where the first stage estimates transductive embedding and the second stage learns two different RNNs for active and inactive users respectively.

Apart from the time-scale difference in temporal dynamics, the simultaneous presence of active and inactive users also presents other modeling and computational challenges. On the one hand, active users lead to long sequences of interactions and high degree nodes in the user-item interaction graph. Existing sequence models, such as RNN models, have some limitations when dealing with *long-range* sequences, due to the difficulty in gradient propagation. Moreover, graph neural network

based models become *computationally inefficient* due to the intensive message passing operations through high-degree nodes introduced by active users. On the other hand, predicting preferences of inactive or new users (also known as the *cold-start* problem) is a challenging few-shot learning problem, where a decision needs to be made given only a few number of observations. To address various challenges imposed by the presence of two types of users, we leverage the different dynamics of these users and propose a **(i) two-time-scale (2TS) model** and a **(ii) two-stage training algorithm**.

2TS model. In our model (Fig. 1), we partition the users into two sets: active users with many observed interactions and inactive users with very few or zero observed interactions. Our 2TS model will update the embeddings of active users and inactive users by two RNNs with independent parameters, in order to respect the two-time-scale nature. Moreover, the initial embeddings of inactive users is represented by a common embedding ψ^0, which is *shared* across all inactive users. Therefore, the overall model for inactive users is an *inductive* model, in the sense that the learned model can be applied to unseen users. In contrast, the initial embedding of each active user is a *user-specific* embedding ϕ_u, which is also called *transductive* embedding. Such embeddings are non-parameteric and very expressive, which can better express users with long history.

Two-stage training. In stage 1, we will first learn transductive user embeddings ϕ_u and transductive item embeddings x_i using a classical collaborative filtering method. Then we will fix these learned transductive embeddings, and then in stage 2, we will learn the parameters of the two RNNs and a common initialization ψ^0 for inactive users. It is notable that we do not use all interaction data for active users to learn the RNN, since the transductive embeddings ϕ_u of active users have already encoded the information of their history. Instead, for each active user, we only use a small number of last clicked items to learn the adaptation, which improves the efficiency of the training process.

The proposed 2TS model and the two-stage training algorithm lead to a few advantages:

- **Bias-variance trade-off.** The expressiveness of transductive model can better leverage the sufficient data points observed from active users to capture their long term history. However, the number of interactions for inactive users is small, making the estimation of their transductive embeddings inaccurate. Our 2TS model let the inactive users share an initial embedding and an inductive model to allow better generalization. The differential use of transductive and inductive embeddings allows 2TS to achieve a good bias-variance trade-off for both active and inactive users.
- **Encode long-range sequence.** The transductive embeddings ϕ_u for active users are user-specific vectors, so they can memorize the user long-range history during the training, without suffering from the difficulty of gradient propagation. The RNN on top of these transductive embeddings is only used for adaptation to recently engaged new items.
- **Computational efficiency.** The efficiency of our method on large-scale problems mainly comes from two designs in the algorithm. First, stage 1 learns the transductive embeddings of active users and items, which contains a large number of parameters. However, it is fast since it does not involve any deep neural components and the loss is simply a convex function. Second, stage 2 only learns the RNNs which contain a small number of parameters, and the RNN for active users is only trained on a few last engaged items, which cuts off the long sequences. Experimentally, our method reveals to be much more efficient than the baselines on large-scale datasets.

We applied our method to two large-scale benchmark datasets, compared comprehensively to various models, spanning a diverse set of categories, and showed that the our method is advantageous in terms of both accuracy and efficiency.

2 Setting

We focus on a **sequential recommendation** setting, where user-item interactions arrive sequentially, and the goal is to predict future interactions based on historical interactions. Consider a set of users $u \in \mathcal{U} := \{1, 2, \cdots, U\}$ and items $i \in \mathcal{I} := \{1, \cdots, I\}$ in a system. We record the interaction between users and items as an ordered sequence $\mathcal{D}$ where the t-th interaction between some user u and item i is denoted as $e_t := (u_t, i_t)$. Then the history of the system at time t is a subset of $\mathcal{D}$ denoted as $\mathcal{D}(t) := \{(u_\tau, i_\tau) : \tau \leq t\}$. We can also slice the interaction data $\mathcal{D}$ according to users, and obtain a sequence of items engaged by user u as $\mathcal{D}_u := \{(u_t, i_t) : (u_t, i_t) \in \mathcal{D} \wedge (u_t = u)\}$. Similarly, the history of a user u before time t can be defined as $\mathcal{D}_u(t) := \{(u_\tau, i_\tau) : (u_\tau, i_\tau) \in \mathcal{D} \wedge (u_\tau = u) \wedge (\tau \leq t)\}$. Besides the sequential representations, we can also aggregate the interactions across time t to obtain an aggregated interaction matrix $A \in R^{U \times I}$, where $A_{ui} = 1$ if $(u_t, i_t) \in \mathcal{D}$ for any t

and 0 otherwise. Furthermore, we will also overload the notation, and use $(u, v) \in A$ to denote the non-zero entries in A, and $|A|$ to denote the number of nonzeros entries in A.

Then the sequential recommendation task is formally defined as predicting which item $i \in \mathcal{I}$ a user u will engage with, when her historical behavior $\mathcal{D}_u(t)$ and the system data $\mathcal{D}(t)$ are given up to time t.

We want to learn a set of user embeddings denoted by $\{\boldsymbol{\phi}_u \in \mathbb{R}^d : u \in \mathcal{U}\}$ and item embeddings denoted by $\{\boldsymbol{x}_i \in \mathbb{R}^d : i \in \mathcal{I}\}$, such that the utility of item i to user u is defined as $r(u, i) := \boldsymbol{\phi}_u^\top \boldsymbol{x}_i$. Furthermore, we will allow the user embeddings to evolve over time, and denote them by $\{\boldsymbol{\phi}_u(t) \in \mathbb{R}^d : u \in \mathcal{U}, t = 1, \ldots, |\mathcal{D}|\}$. We will denote the embedding of a user right before the t-th interaction as $\boldsymbol{\phi}_u(t^-)$, and the corresponding history leading to $\boldsymbol{\phi}_u(t^-)$ as $\mathcal{D}(t^-)$. Given an interaction $(u_t, i_t) \in \mathcal{D}(t^-)$, we will measure the performance of a model using a softmax loss:

$$\ell_t := \ell((u_t, i_t), \boldsymbol{\phi}_u(t^-), \{\boldsymbol{x}_i\}) = \log\Big[\sum\nolimits_{j \in \mathcal{O}(u_t, i_t)} \exp(\boldsymbol{\phi}_{u_t}^\top(t^-)\boldsymbol{x}_j)\Big] - \boldsymbol{\phi}_{u_t}^\top(t^-)\boldsymbol{x}_{i_t}. \tag{1}$$

where the set $\mathcal{O}(u_t, i_t)$ is the set of items displayed to user u_t when he/she clicks on the item i_t, and $i_t \in \mathcal{O}(u_t, i_t)$. In our model, we will divide the users into two sets $\mathcal{U}_{act}$ and $\mathcal{U}_{in}$ by the *numbers of observed interactions* T in the training set. More precisely, with a threshold $t^* \in \mathbb{N}$, we consider users with more than t^* observed interactions as active users $\mathcal{U}_{act}$, and those with less than t^* observed interactions as inactive users $\mathcal{U}_{in}$, which include new users. Similarly, we can divide the set of interactions according to active and inactive user as $\mathcal{D}_{act} = \{(u_t, i_t) : (u_t, i_t) \in \mathcal{D} \wedge u_t \in \mathcal{U}_{act}\}$ and $\mathcal{D}_{in} = \{(u_t, i_t) : (u_t, i_t) \in \mathcal{D} \wedge u_t \in \mathcal{U}_{in}\}$ In our later section, we will treat the evolution of embeddings for active and inactive users differently using a two-time-scale model.

3 Why Two-time-scale models?

Apart from the information theory view in the introduction, we will provide another intuition on the two-time-scale model for user embeddings from an online learning point of view. For simplicity of exposition, we will assume that item embeddings $\{\boldsymbol{x}_i\}$ are already learned and fixed. Then as a user-item interaction (u_t, i_t) arrives in a sequential fashion, the user embedding can be updated according to an online gradient descent algorithm for the loss in (1) as

$$\boldsymbol{\phi}_{u_t}(t) = \boldsymbol{\phi}_{u_t}(t^-) - \eta_t \frac{\partial \ell_t}{\partial \boldsymbol{\phi}_{u_t}(t^-)} = \boldsymbol{\phi}_{u_t}(t^-) - \eta_t\Big(\sum\nolimits_{j \in \mathcal{O}(u_t, i_t)} p_j\, \boldsymbol{x}_j - \boldsymbol{x}_{i_t}\Big) \tag{2}$$

where $p_j := \frac{\exp(\boldsymbol{\phi}_{u_t}^\top(t^-)\boldsymbol{x}_j)}{\sum_{j \in \mathcal{O}(u_t, i_t)} \exp(\boldsymbol{\phi}_{u_t}^\top(t^-)\boldsymbol{x}_j)}$ is the current model's predicted probability of clicking item j. We can see that the magnitude of the gradient is related to the discrepancy between the model prediction and the actual observation. In particular, if the model is very accurate, that is $p_{i_t} = 1$ placing probability 1 on the clicked item i_t, the gradient $\sum_{j \in \mathcal{O}(u_t, i_t)} p_j\, \boldsymbol{x}_j - \boldsymbol{x}_{i_t} = p_{i_t}\boldsymbol{x}_{i_t} - \boldsymbol{x}_{i_t} = \mathbf{0}$. In this case no update to $\boldsymbol{\phi}_{u_t}$ will be made. More specifically, from the update we can observe different temporal dynamics for active and inactive users:

- Only the embedding for user u_t will be updated when we observe an interaction (u_t, i_t). Therefore, active users will have lots of gradient updates, and their embeddings will be learned sufficiently well and converge close to the true embeddings as learned in batch setting. In contrast, for inactive users, there will be very few observations and gradient updates to the corresponding user embeddings. In this case, the embedding will still be far from the true embedding as is learned in batch setting.
- Thus, when a new interaction associated with an active user u_t is observed, the discrepancy between the model prediction and actual observation will be small, making the gradient and embedding change very small. In other words, a new interaction provides diminishing amount of information if we already have observed lots of information about this user.
- In contrast, when a new interaction associated with an inactive user u_t is observed, the discrepancy between model prediction and the actual observation will be large, making the gradient and embedding change very large. In other words, each new interaction provides us a lot of information about this user, and we want to carefully model the contribution of each interaction to the evolution of the user embedding.

In summary, if we assume that user embeddings are evolving in an online learning fashion, the evolution of these embeddings will occur roughly in two time scales. For active users, the embeddings evolve in **slow time scale** and remain more or less unchanged given a few new observations; while for inactive users, the embeddings evolve in **fast time scale** and every interaction counts.

This two-time-scale phenomenon implies that the embeddings for active users could be estimated well from a batch setting, and only small sequential updates are needed afterwards; in contrast, for inactive users, their embeddings can change a lot, and sequential update models will be playing an important role in estimating their embeddings. This intuition motivates us to design a two-time-scale model to treat active and inactive users differently, and a two-stage method to train our model.

4 METHODOLOGY

In this section, we will describe our parameterization of two-time-scale (2TS) model and a two-stage method for learning this model efficiently and effectively. Instead of following exactly the online gradient decent model in the previous section, we will use a more general framework, the meta-learning framework (Ravi & Larochelle, 2016), to model the update dynamics of user embeddings. In particular, we will use two different RNNs to model the dynamics of active and inactive users respectively, and hence the name for the two-time-scale model. Instead of training the RNN for active user from long sequences directly, we will avoid directly modeling long sequences using RNNs, and leverage the slow evolving nature of the embeddings of active users. More precisely, we will first perform a generalized factorization of an aggregated interaction matrix A to obtain *transductive* user and item embeddings; and then use these transductive user embeddings as the initial embeddings during the training of our RNNs. The intuition is that the transductive embeddings obtained from the first stage will already reflect the long term behavior of the active users, and hence will provide very good initialization for later sequential recommendation and updates; Meanwhile, the transductive embeddings for items obtained from the first stage will provide very good basis for learning the RNN for the inactive users who typically only have few interactions.

In real life system, we can perform the active and inactive user partition periodically, and retrain the transductive embedding and the two-time-scale RNNs. Within each cycle, we will run the two-time-scale RNN model for sequential recommendation until the next cycle starts. The details of our models and algorithms are explained below.

4.1 TWO-TIME-SCALE MODEL

Our two-time-scale model consists of two RNNs, one for active users and the other for inactive users. They can be thought of as two different online adaptation operators for active and inactive users respectively. Furthermore, these two RNNs will also differ in the way in whether they have user-specific parameters and how they are initialized. Assume for the moment, we have already learned an initial set of transductive user embeddings $\left\{\boldsymbol{\phi}_u \in \mathbb{R}^d : u \in \mathcal{U}\right\}$ and item embeddings $\left\{\boldsymbol{x}_i \in \mathbb{R}^d : i \in \mathcal{I}\right\}$, which will be explained in later two-stage training methods.

The RNN for inactive or new users is purely inductive in the sense that it does not contain user-specific parameters. We will **not** use user-specific transductive embeddings $\boldsymbol{\phi}_u$ in this RNN. Instead, the RNN will start with a learned **common initialization** $\boldsymbol{\psi}^0 \in \mathbb{R}^d$, and then be updated by RNN-based adaptation operators every time a new interaction is observed. More precisely, for an inactive user who clicks on items $(i_1, \cdots, i_T)$ sequentially, his/her embeddings will be updated sequentially as:

$$\textbf{inactive users } u \in \mathcal{U}_{in}\textbf{:} \quad \boldsymbol{\psi}_u^0 = \boldsymbol{\psi}^0; \quad \boldsymbol{\psi}_u^{t+1} = \texttt{RNNcell}_{\Theta_1}(\boldsymbol{\psi}_u^t, \boldsymbol{x}_{i_t}). \tag{3}$$

For active users, we will view their transductive embeddings as the memories for their long-term history. Different from inactive users, the initial embeddings in the RNN for active users will be set to be the user-specific transductive embeddings $\boldsymbol{\phi}_u$. Then the RNN-based adaptation operator will update this initial transductive user embeddings if there are more interacted items:

$$\textbf{active users } u \in \mathcal{U}_{act}\textbf{:} \quad \phi_u^0 = \phi_u; \quad \phi_u^{t+1} = \texttt{RNNcell}_{\Theta_2}(\phi_u^t, x_{i_t}). \tag{4}$$

The inactive and active user models in Eq. 3 and Eq. 4 constitute our two-time-scale model (2TS model). The model consists of several set of parameters, including the transductive user embedding $\left\{\boldsymbol{\phi}_u \in \mathbb{R}^d : u \in \mathcal{U}\right\}$ and item embeddings $\left\{\boldsymbol{x}_i \in \mathbb{R}^d : i \in \mathcal{I}\right\}$, the two sets of RNN parameters Θ_1 and Θ_2 respectively, and the additional common initialization $\boldsymbol{\psi}^0$ for the inactive user RNN. These parameters will be learned differently using a two-stage method described in the next section.

4.2 TWO-STAGE TRAINING

Our training method for the two-time-scale model consists of two stages. In **stage 1**, we will learn the transductive embeddings for all users and items from observed interactions $\mathcal{D}$. In **stage 2**, we will the learn the parameters of the two RNN. In summary, the blue-colored transductive embeddings for active users $\{\boldsymbol{\phi}_u : u \in \mathcal{U}_{act}\}$ and items $\{\boldsymbol{x}_i : i \in \mathcal{I}\}$ are learned in stage 1 and fixed in stage 2. The red-colored components that include the common initialization $\boldsymbol{\psi}^0$ for inactive users and the RNN parameters Θ_1 and Θ_2 are learned in stage 2. More specifically,

Stage 1. We will first ignore the sequential ordering of the interactions, and use the aggregated interaction matrix A to learn the transductive user and item embeddings:

$$\{\boldsymbol{\phi}_u\}, \{\boldsymbol{x}_i\} = \arg\min_{\{\boldsymbol{\phi}_u\},\{\boldsymbol{x}_i\}} \frac{1}{|A|} \sum_{(u,i)\in A} \log\left[\sum_{j\in\mathcal{O}(u,i)} \exp(\boldsymbol{\phi}_u^\top \boldsymbol{x}_j)\right] - \boldsymbol{\phi}_u^\top \boldsymbol{x}_i, \tag{5}$$

where the set $\mathcal{O}(u,i)$ is the set of items displayed to user u when he/she clicks on the item i. Typically this information is not given, so we could randomly sample p items as the non-clicked items, and use these p items together with the clicked item i to constitute the offer set $\mathcal{O}(u,i)$.

Stage 1 training is similar to collaborative competitive filtering (CCF) in Yang et al. (2011). It is efficient since the objective is a simple function. Besides, since the objective is convex in either $\boldsymbol{\phi}_u$ or $\boldsymbol{x}_i$, it is easier to obtain the global optimum and the results are affected less by the initialization.

Since active users have many interactions, their transductive embeddings are learned well and lead to good predictions in heldout data. However, for inactive users with fewer observed interactions, the learned transductive embeddings could be overfitted. Thus, in the next stage of training RNNs, we will re-use the transductive embeddings for active users but will discard those for inactive users. Furthermore, the learned item embeddings $\{\boldsymbol{x}_i\}$ will also be used in the next stage of training.

Stage 2. We will train the parameters of the two RNNs using the following loss functions:

$$\mathcal{L}_{in}(\Theta_1, \boldsymbol{\psi}^0) := \frac{1}{|\mathcal{D}_{in}|} \sum_{(u_t,i_t)\in\mathcal{D}_{in}} \log\left[\sum_{j\in\mathcal{O}(u_t,i_t)} \exp(\boldsymbol{\psi}_{u_t}^\top \boldsymbol{x}_j)\right] - \boldsymbol{\psi}_{u_t}^\top \boldsymbol{x}_{i_t}, \tag{6}$$

$$\mathcal{L}_{act}(\Theta_2) := \frac{1}{|\mathcal{D}_{act}^K|} \sum_{(u_t,i_t)\in\mathcal{D}_{act}^K} \log\left[\sum_{j\in\mathcal{O}(u_t,i_t)} \exp(\boldsymbol{\phi}_{u_t}^\top \boldsymbol{x}_j)\right] - \boldsymbol{\phi}_{u_t}^\top \boldsymbol{x}_{i_t}, \tag{7}$$

where $\mathcal{D}_{in} \subset \mathcal{D}$ contains the training data for inactive users, and $\mathcal{D}_{act}^K \subset \mathcal{D}_{act} \subset \mathcal{D}$ contains the last K interactions for each active user observed in the training set. We note that the embedding for each user $\boldsymbol{\phi}_{u_t}$ is updated sequentially using the corresponding RNNs in Eq. 4 and Eq. 3 respectively. Furthermore, for an active user who has more than t^* interactions, we will only use the last K interaction of the user to train the RNN. There are two reasons. First, it allows us to avoid the direct use of RNN for long sequence modeling, which is inefficient. Second, the transductive embeddings for active users has already encoded most information from these users and only small online adaption is needed to boost the performance further.

Furthermore, during the entire training process of the two RNNs, the transductive embeddings of the items $\{\boldsymbol{x}_i\}$ are fixed. This further increases the efficiency of the training in stage 2. In fact, the transductive item embeddings learned in stage 1 have already provided good representations for the items. Recent theoretical studies also showed that learning item embeddings as a first stage can drastically improve the sample complexity of a model for few shot learning in a new task (Du et al., 2020). More specifically, assume the number of interactions for an inactive user is n, then the sample complexity for learning a recommendation model for this user can be reduced from $\mathcal{O}(|\mathcal{I}|/n)$ to $\mathcal{O}(d/n)$, where the number of items $|\mathcal{I}|$ can be much larger than the embedding dimension d.

4.3 IMPLEMENTATION DETAILS

We present two implementation details that are essential for the performance of our model in the experiments but less relevant to the main ideas of this paper.

2TS-Plus. We propose a variant of our 2TS model called 2TS-Plus. The only difference from 2TS is that, we replace the user embeddings $\boldsymbol{\phi}_{u_t}$ in 2TS by

$$\widehat{\boldsymbol{\phi}_{u_t}} \leftarrow W^\top \begin{bmatrix} \boldsymbol{\phi}_{u_t} \\ \boldsymbol{x}_{i_{t-1}} \end{bmatrix}, \tag{8}$$

where $\boldsymbol{x}_{i_{t-1}}$ is the item embedding of the user u_t's most recently clicked item i_{t-1}, and W is a learnable weight matrix. In summary, 2TS-Plus explicitly incorporate the information of the 'last clicked item' to compute the user embeddings. Our experiments show that this can consistently improve the performance. In fact, we incorporate the 'last clicked item' because we found that the baseline JODIE (Kumar et al., 2019) performs particularly well on lastfm-1K. Thus we looked into their implementation and found that it is the 'last clicked item' that helps with the performance.

Features. In many datasets, item features $\{\boldsymbol{f}_i\}_{i\in\mathcal{I}}$ are provided. In the experiments, we concatenate the transductive item embeddings $\boldsymbol{x}_i$ with the feature-based embedding, $g_\varphi(\boldsymbol{f}_i)$, where g_φ a simple network with parameter φ. The concatenation $[\boldsymbol{x}_i^\top, g_\varphi(\boldsymbol{f}_i)^\top]^\top$ will be used as the item embeddings, and both transductive embeddings $\boldsymbol{x}_i$ and the parameters φ are learned in stage 1.

5 Related Work

The stage 1 training of our method is most related to methods based on matrix factorization (MF) (Koren et al., 2009), where latent user and item vectors are learned to recover historical user-item interactions. Many methods for MC are collaborative filtering (CF)-based (Koren et al., 2009).

Neural models for recommendation broadly fall into three categories: graph-based models, deep sequence models, and dynamic graph models.

Graph-based models. Monti et al. (2017) first studied recommender systems with graph neural network (GNN). Berg et al. (2017) proposed graph convolutional matrix completion (GCMC) which applies a GNN on the user-item bipartite graph to learn user and item embeddings. It is a transductive model, as it learns user/item-specific embeddings. PinSage (Ying et al., 2018) and IGMC (Zhang & Chen, 2020) are recently proposed inductive GNN-based recommender system. Despite showing promising results, existing GNN-based recommender systems usually have poor scalability, due to their expensive neighborhood sampling procedure for performing graph convolution.

Dynamic graph models. Based on the idea that both users and items will evolve over time via temporal interactions, Dai et al. (2016); Farajtabar et al. (2017); Kumar et al. (2019); Goyal et al. (2020); Sankar et al. (2020) take the graph temporal evolution into modeling design, along with graph topology. However, most existing dynamic graph approaches cannot scale to large interaction graphs. JODIE (Kumar et al., 2019) proposed a batching method to make the training process more efficient.

Deep Sequence models. RNN (Hidasi et al., 2015; Jannach & Ludewig, 2017) and LSTM (Devooght & Bersini, 2017; Chen et al., 2019) based deep models are widely used in sequential recommendation. Other methods based on attention models (Zhou et al., 2018; 2019) have also been explored. However, these models are still have difficulties in leveraging the information contained in states located far into the past due to gradient propagation issues (Pascanu et al., 2013). Several recent advances are proposed to deal with long-range sequences (Tang et al., 2019; Pi et al., 2019)

Cold-start problem. Traditional approaches to address cold-start problems include content filtering, hybrid models, and presenting a few items to new users. Vartak et al. (2017) proposed a meta-learning perspective for the item cold-start problem, where recommending new items to users is formulated as learning a learning algorithm. Lee et al. (2019) proposed to use meta-learning to estimate new user's preferences with a few consumed items. Bose et al. (2019) also proposed Meta-Graph to perform few-shot link prediction across multiple graphs. Wu et al. (2020) proposed to compute embeddings for new users using the embeddings of active users, via an attention-based model. They share the same idea of splitting the users into two sets, but their main target is the matrix factorization problem and do not consider temporal evolution of user embeddings and their two-time-scale difference.

6 Experiments

Table 2: Dataset Statistics

Dataset	#users	#items	#interactions
lastfm-1K	1,000	1,000	1,293,103
ML-25M	162,538	59,048	24,999,849
Taobao	987,975	4,111,798	96,678,667

To validate the performance of 2TS model, we conduct experiments on three public datasets, two of which are the largest recommendation benchmark datasets that are closer to industrial scenario, making our results more solid and convincing. By comparing to a diverse set of state-of-the-art (SOTA) methods, we demonstrate the scalability and accuracy of our 2TS model. The detailed configurations of 2TS are in Appendix B.

Dataset. We consider 3 public datasets: Taobao dataset (Pi et al., 2019), MovieLens-25M (ML-25M) (Harper & Konstan, 2015), and lastfm-1K (Celma, 2010). Dataset statistics are summarized in Table 2. Taobao and ML-25M are large-scale. Especially, Taobao contains about 100 million interactions. The results on these two datasets are much more convincing. All datasets provide user-item interaction data where for each interaction, user ID, item ID, time-stamp, and item feature are given. For ML-25M, we ignore the ratings and simply use it as interaction data. For each dataset, we sort the interactions by time-stamp, and use the first 70% as the training set, the following 10% as the validation set, and the last 20% as the test set. All reported results are estimated on the test set.

Baselines. We compare 2TS with 9 SOTA models spanning 4 categories: (i) **CF-based methods**: Collaborative competitive filtering (CCF) (Yang et al., 2011) is an advanced CF model which takes into account the context of user's choice. It is a simple yet very effective method. (ii) **GNN-based models**: GCMC (Berg et al., 2017) is a SOTA graph-based architecture for recommendation, and GCMC-SAGE is its stochastic variant which is more efficient. (iii) **Deep sequence models**: SumPooling is a simple yet effective baseline that is widely used in industry. GRU4REC (Hidasi et al., 2015) is a representative of RNN-based models. DIEN (Zhou et al., 2019) is an advanced attention-based sequence model. MIMN (Pi et al., 2019) is a memory-enhanced RNN architecture to better capture long sequential user behaviors. It is a strong baseline and the authors come from the team that publicized Taobao dataset. (iv) **Dynamic graph models**: JODIE (Kumar et al., 2019) and dynAERNN (Goyal et al., 2020) are two dynamic temporal graph approaches which learn graph node dynamic embeddings via temporal modeling. Note that the released implementations of some baselines are not scalable to large-scale datasets such as Taobao, so we made modifications to make sure baselines are runnable on these datasets (See Appendix A).

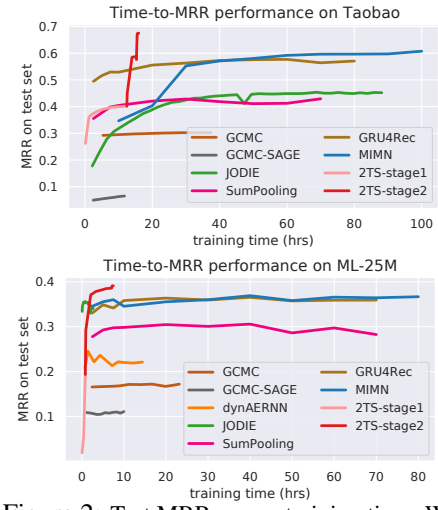


Figure 2: Test MRR versus training time. We compare the convergence speeds of different models in wall-clock time. Run-time on lastfm-1K is not showed because it is small-scale.

Evaluation Metric. We measure the performance of different models in terms of the mean reciprocal rank (MRR), the average of the reciprocal rank, and recall@10, which is the fraction of interactions in which the ground truth item is ranked in the top 10. For both metrics, higher values are better. For

Table 1: Overall performance for interaction prediction. The best method in each column is colored blue and best method among the baselines is colored light blue. Note that the lastfm-1K dataset is *much smaller* than Taobao and ML-25M. We run the experiments on lastfm-1K just to show that even when the baselines are sufficiently optimized on this small dataset, 2TS and 2TS-Plus can still perform better. However, the real advantages of our models are for large-scale problems.

	method	Taobao		ML-25M		lastfm-1K	
		MRR	Rec@10	MRR	Rec@10	MRR	Rec@10
CF-based	CCF	0.402	0.621	0.193	0.392	0.051	0.116
GNN-based models	GCMC	0.303	0.542	0.171	0.378	0.081	0.129
	GCMC-SAGE	0.147	0.357	0.109	0.264	0.168	0.208
Deep sequence models	SumPooling	0.415	0.664	0.302	0.591	0.071	0.180
	GRU4REC	0.546	0.777	0.364	0.657	0.152	0.344
	DIEN	0.605	0.834	0.356	0.638	0.100	0.213
	MIMN	0.607	0.828	0.363	0.653	0.115	0.261
Dynamic graph models	dynAERNN	-oom-	-oom-	0.249	0.509	0.021	0.038
	JODIE	0.454	0.680	0.354	0.634	0.176	0.325
Our method	**2TS**	**0.669**	**0.844**	**0.391**	**0.679**	**0.151**	**0.332**
	2TS-Plus	**0.680**	**0.844**	**0.409**	**0.691**	**0.203**	**0.369**
Improvement to best baseline	**2TS**	10.2%	1.2%	7.4%	3.3%	-	2.2%
	2TS-Plus	12.0%	1.2%	12.4%	5.2%	15.3%	13.5%

every interaction, the ranking of ground truth item is calculated with respect to 500 items where the other 499 negative items are randomly sampled from the set of all items.

Overall Performance (Table 1)**.** We summarize the overall performance in terms of MRR and Rec@10 in Table 1. On the large-scale datasets Taobao and ML-25M, our model 2TS and 2TS-Plus have consistent and significant improvements compared to all baselines. Especially, in terms of MRR, an improvement of at least 7.4% and 12.0% are achieved by 2TS and 2TS-Plus, respectively. On the small dataset lastfm-1K, we found that the information of the 'last clicked item' is particularly useful on this dataset. Since both JODIE and 2TS-Plus have explicitly modeled the 'last clicked item', they achieve the best results on lastfm-1K, while 2TS-Plus is still 15.3% better in terms of MRR.

Scalability & Efficiency (Fig. 2)**.** To compare the training efficiency, we evaluate different models' intermediate checkpoint performances on the test set. Fig. 2 shows that the performance of 2TS increases fast. 2TS's first stage transductive training is efficient and provides an effective initialization for the second stage inductive training. Besides, though not revealed from the figures, the training time of 2TS for each epoch is much smaller than the baselines.

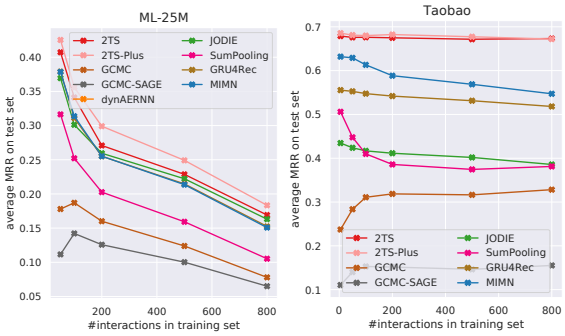


Figure 3: Test MRR in different segments of users.

Performance for different users. Fig. 3 shows the MRR performances averaged over users with different numbers of observed interactions (the more interactions the more active the user is). 2TS and 2TS-Plus lead to consistent improvements over the entire range of users. We note that in ML-25M, most models achieved higher MRR for inactive users than active users. This is *not* counter-intuitive since in the movielens dataset inactive users very often click on popular movies, making their behaviors more predictable than active users.

Two-time-scale behavior (Fig. 4)**.** Recall 2TS has two RNNs. One updates the embeddings of inactive users and the other updates those of active users. Given the learned RNNs, on the test set, we compute the changes of user embeddings in 2-norm as the user interacts with more items (Fig. 4). The behavior aligns with our intuition. Embeddings of inactive users are changed faster and those of active users are more static.

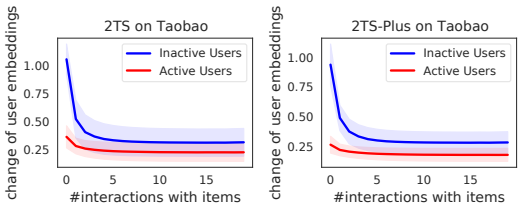


Figure 4: Two-time-scale behavior of user embeddings.

Ablation Study (Table 3)**.** To show the effectiveness of the two-stage training, we compare 2TS to 2TS-SingleStage, who has exactly the same architecture as 2TS but all parameters are trained together in a single stage. 2TS-SingleStage performs worse, which means the transductive embeddings are learned very well in Stage 1, benefit from the convexity and easiness of optimization. Besides, the training of 2TS-SingleStage is a lot slower. Furthermore, to show the effectiveness of the 2TS model design, we compare it to 2-GRU, which applies two different RNNs to inactive and active users, but there are no transductive user embeddings for active users.

Table 3: Ablation study.

method	ML-25M		Taobao	
	MRR	Rec@10	MRR	Rec@10
2TS	0.391	0.679	0.669	0.894
2TS-SingleStage	0.373	0.636	0.420	0.533
2-GRU	0.354	0.643	0.597	0.791

7 Conclusion

We proposed to learn two-time-scale representation (2TS) for large recommendation systems to explicitly address the learning efficiency for different user dynamics. We also proposed a two-stage training schema, to leverage the transductive embedding as the inductive model initialization for active users. We evaluated 2TS on large scale recommendation ranking task, and showed that 2TS is able to outperform several class of state-of-the-art methods including deep sequence models, graph neural network, and dynamic graphs. Our results show that, designing different representation to capture diverse interaction dynamics are desirable. Future work includes separating and updating user dynamic in an algorithmic way.

REFERENCES

Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

Avishek Joey Bose, Ankit Jain, Piero Molino, and William L Hamilton. Meta-graph: Few shot link prediction via meta learning. *arXiv preprint arXiv:1912.09867*, 2019.

O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. Generative adversarial user model for reinforcement learning based recommendation system. In *International Conference on Machine Learning*, pp. 1052–1061. PMLR, 2019.

Hanjun Dai, Yichen Wang, Rakshit Trivedi, and Le Song. Deep coevolutionary network: Embedding user and item features for recommendation. *arXiv preprint arXiv:1609.03675*, 2016.

Robin Devooght and Hugues Bersini. Long and short-term recommendations with recurrent neural networks. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, pp. 13–21, 2017.

Simon S Du, Wei Hu, Sham M Kakade, Jason D Lee, and Qi Lei. Few-shot learning via learning the representation, provably. *arXiv preprint arXiv:2002.09434*, 2020.

Mehrdad Farajtabar, Yichen Wang, Manuel Gomez-Rodriguez, Shuang Li, Hongyuan Zha, and Le Song. Coevolve: A joint point process model for information diffusion and network evolution. *The Journal of Machine Learning Research*, 18(1):1305–1353, 2017.

Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, 187:104816, 2020.

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

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

Dietmar Jannach and Malte Ludewig. When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pp. 306–310, 2017.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1269–1278, 2019.

Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. Melu: Meta-learned user preference estimator for cold-start recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1073–1082, 2019.

Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 3700–3710, 2017.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318, 2013.

Qi Pi, Weijie Bian, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. Practice on long sequential user behavior modeling for click-through rate prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2671–2679, 2019.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pp. 519–527, 2020.

Jiaxi Tang, Francois Belletti, Sagar Jain, Minmin Chen, Alex Beutel, Can Xu, and Ed H. Chi. Towards neural mixture recommender for long range dependent user sequences. In *The World Wide Web Conference*, pp. 1782–1793, 2019.

Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. A meta-learning perspective on cold-start recommendations for items. In *Advances in neural information processing systems*, pp. 6904–6914, 2017.

Qitian Wu, Hengrui Zhang, and Hongyuan Zha. Inductive relational matrix completion. *arXiv preprint arXiv:2007.04833*, 2020.

Shuang-Hong Yang, Bo Long, Alexander J Smola, Hongyuan Zha, and Zhaohui Zheng. Collaborative competitive filtering: learning recommender using context of user choice. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pp. 295–304, 2011.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 974–983. ACM, 2018.

Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ByxxgCEYDS.

Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1059–1068, 2018.

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

A BASELINE SPECIFICATION

Some of the compared baseline methods are not directly scalable to large scale interaction graphs, or originally designed for ranking task. To make the baselines runable on large and sparse graphs and comparable to our proposed method, we have made a few adaptions.

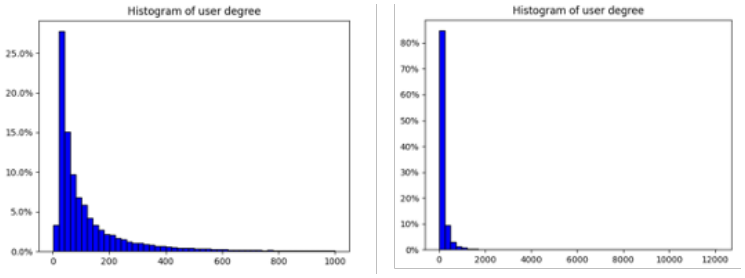
- **JODIE**: we made the following adaptions: 1) replaced the static one-hot representation to 64-dim embeddings because the number of nodes are large; 2) represent the category / tag categorical feature via a learnable embedding table; 3) used triplet loss with random negative example rather than original MSE loss, which empirically show improvements.
- **dynAERNN**: These two methods are originally designed for discrete graph snapshots, while our focused tasks are continues interaction graphs. We manually transformed the interactions graph into 10 snapshots, with equal edge count increments.

  For the downstream ranking task, we followed the evaluation method used in dySat (Sankar et al., 2020): after the node embeddings are trained, we train a logistic regression classifier to predict link between a pair of user / item node embedding using Hadmard operator. The logits on test set are used for ranking. For Taobao dataset, we are not able to get reslt of dynAERNN given the memory constraint.
- **sum-pooling & DIEN & MIMN**: We changed the data batching from per-user based to per-example based, to better adapt for time ordered interaction data.
- **GCMC & GCMC-SAGE**: We changed the loss function from Softmax over different ratings to Softmax over [true item, 10 random selected items], to better adapt to the ranking task.

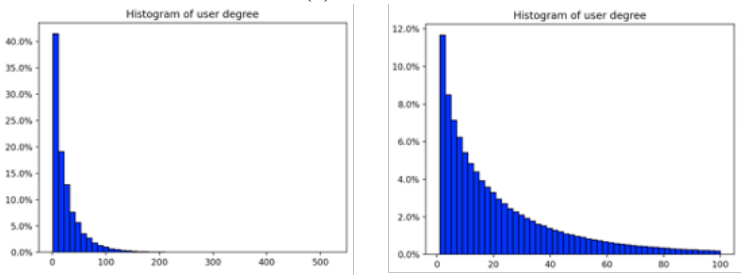
B CONFIGURATION OF 2TS AND 2TS-PLUS

We will present some important hyperparameters chosen for the 2TS and 2TS-Plus in the experiments.

- Threshold $t^* \in \mathbb{N}$: Users with more than t^* interactions observed in the training set are considered active users, and those with less than or equal to t^* observed interactions are considered inactive users. To choose the threshold t^* we can first observe the distribution of number of interactions of the users (Fig. 5). For both 2TS and 2TS-Plus, we choose $t^* = 100$ for both Taobao and ML-25M datasets.
- Last K interactions for active users: In Stage 2 training, for each active users, we only use the last K interactions to train the RNN. The number K is chosen to be either 10 or 20, and they actually give similar performances.
- Embedding dimension: Both user embeddings and item embeddings are set to be of dimension 64. If the item contains features, we will use another 64 dimension for the feature embedding.
- Learning rate: the learning rate for training 2TS and 2TS-Plus is searched over $1e-3$ and $1e-4$ only. Most of the time $1e-4$ works better. The optimizer is Adam.



(a) ML-25M



(b) Taobao

Figure 5: Distribution of the number of observed interactions in the training dataset.