Billion-scale Recommendation with Heterogeneous Side Information at Taobao

Andreas Pfadler*, Huan Zhao†, Jizhe Wang*, Lifeng Wang*, Pipei Huang* and Dik Lun Lee†
*Alibaba Group, Beijing&Hangzhou, China
†The Hong Kong University of Science and Technology, Hong Kong SAR, China
*{andreaswernerrober,jizhe.wjz,tiechou,pipei.hpp}@alibaba-inc.com, †{hzhaoaf,dlee}@cse.ust.hk

***Abstract*—In recent years, embedding models based on skip-gram algorithm have been widely applied to real-world recommendation systems (RSs). When designing embedding-based methods for recommendation at Taobao, there are three main challenges: scalability, sparsity and cold start. The first problem is inherently caused by the extremely large numbers of users and items (in the order of billions), while the remaining two problems are caused by the fact that most items have only very few (or none at all) user interactions. To address these challenges, in this work, we present a flexible and highly scalable Side Information (SI) enhanced Skip-Gram (SISG) framework, which is deployed at Taobao. SISG overcomes the drawbacks of existing embedding-based models by modeling user metadata and capturing asymmetries of user behavior. Furthermore, as training SISG can be performed using any SGNS implementation, we present our production deployment of SISG on a custom-built word2vec engine, which allows us to compute item and SI embedding vectors for billion-scale sets of products in a join semantic space on a daily basis. Finally, using offline and online experiments we demonstrate the significant superiority of SISG over our previously deployed framework, EGES, and a well-tuned CF, as well as present evidence supporting our scalability claims.**

***Index Terms*—large-scale recommendation, embedding-based methods**

I. Introduction

Recommender Systems (RSs) have been a major driving force behind the online E-commerce business at Taobao, China's largest online consumer-to-consumer (C2C) platform owned by Alibaba. The key role of RS is to provide users with interesting items, i.e., commodities, which can improve both user satisfaction and overall conversion rate. However, Taobao's billion-scale set of users and items present extreme challenges to the deployment of an effective RS solution. To address scalability, the RS platform employs a two-stage pipeline. The first stage is **matching**, and the second is **ranking**. In the matching stage, a candidate set of similar items is obtained for each item that users have interacted with. Then, in the ranking stage, a prediction model, e.g. a deep neural network model [2]–[4], ranks the candidate items for each user according to his or her preferences.

In this paper, we focus on the matching stage, for which the key task is to compute the similarity between items. This process is very important, since only a small number (thousands) of items are selected out of roughly 1 billion items into the candidate set when a user clicks an item. Traditionally, collaborative filtering (CF) methods [10], [12] have been adopted to compute item similarity, which rely on the co-occurrence of two items in user behavior sequences. Recently, researchers have proposed to compute the item similarities based on embedding methods, which represent each item with a low-dimensional vector, i.e., embedding. The key component is built on top of a skip-gram based algorithm from word2vec [14], [15], which has been demonstrated effective for learning word embeddings according to "context information" from sequences of words, i.e., sentences. Likewise, by modeling users' clicked items as sequences, a series of embedding methods have been proposed for recommendation [6], [11], [22], [24], [30].

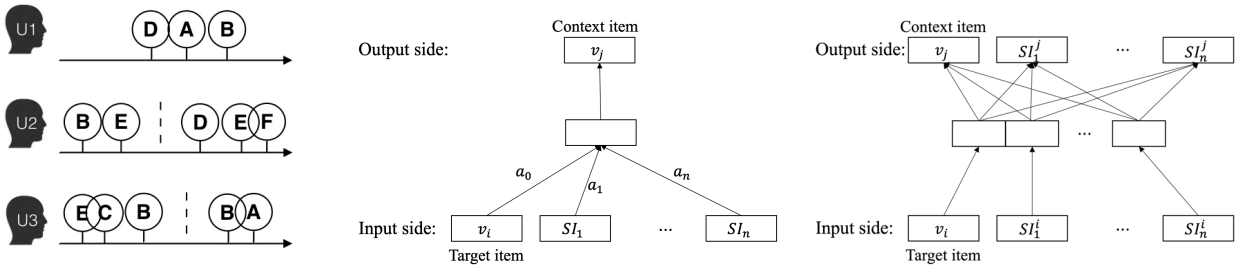
When using embedding methods for recommendation at Taobao, there are three general challenges: scalability, sparsity and cold start. The first problem is inherently caused by the extremely large numbers of users and items, while the remaining two problems are caused by the fact that most items have only very few (or none at all) user interactions. In the literature, to alleviate the sparsity and cold start problems, researchers have tried to utilize side information (SI), e.g., item metadata [28] or social connections among users [25], [27], which can provide supervisions to infer users' preferences when their clicked behaviors are sparse. However, the scalability challenge becomes more critical when incorporating heterogeneous SI since the scale of the total numbers of SIs is the same as that of users and items.

To the best of our knowledge, no existing embedding based methods [6], [11], [22], [24], [30] have been reported to be able to work on the scale that is required at Taobao, i.e. billions of items. Besides, when modeling users' behavior sequences with the skip-gram framework, they ignore the inherent asymmetry exhibited in user behaviors, that is, the probability of a user clicking item B after item A is rarely the same as that of clicking A after B. In our case, since we need to predict which item a user will click based on his/her previous click sequences (see Figure 1(a)), the order of user clicks matters.

We can thus summarize the key problems we faced in our production scenario as follows: *How can we build a matching engine which includes heterogeneous SIs, i.e., item and user metadata, models user behavior asymmetries, provides embed-*

Andreas Pfadler and Huan Zhao contributed equally to this work, and Pipei Huang is the corresponding author.

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00148



(a) User behavior sequences. (b) EGES framework. (c) SISG framework.

Fig. 1. a) User behavior sequences; Different sessions are separated by dashed lines; b) The EGES framework; c) The SISG framework.

dings in a joint semantic space, all the while still being able to scale up to billions of items ? Moreover, how can we make sure that all (possibly billions) embeddings may be computed on a daily basis?

The above considerations have led us to revisit embedding-based approaches for recommendation based on the key component of word2vec: Skip-Gram with Negative Sampling (SGNS) [6], [7], [22]. Based on this well-established approach, we have designed a highly-scalable and flexible SI enhanced Skip-Gram (SISG) framework for performing the recommendation task at Taobao. The key advantage of SISG consists is that training may in principle be implemented using any word2vec [15] implementation. Unfortunately, at the time there were no suitable word2vec implementations available which could support our scalability requirements. We have thus chosen to implement SISG on top of a customized and and distributed word2vec engine, the key components of which are described in this article.

Overall, the spotlights of this paper can be summarized as follows:

- **Flexible handling of user and item SIs.** Our SISG framework is flexible enough to incorporate any type of SI, including item and user metadata, which have been shown to be highly effective for the recommendation task [20], [22], [25], [28], [29]. This flexible design allows SISG to be applied to other recommendation scenarios with simple modifications.
- **Capturing asymmetry of user behaviors.** We customize SISG in its sampling of skip-grams and similarity computation to differentiate the transition probability from one item to another and the other way around. We conduct extensive experiments to demonstrate the effectiveness of capturing the asymmetry underlying user behaviors.
- **Efficient distributed training pipeline.** To deploy the SISG framework in Taobao production environment, where billion-scale sets of users and items are available, we have implemented a customized word2vec engine based on two novel key components: Global training with Adapted Target Negative Sampling (ATNS) and Heuristic Balanced Graph Partition (HBGP). ATNS can greatly decrease the communication cost across workers when training SISG and scale to arbitrary graph sizes. HBGP is designed to allocate computational resources uniformly. Based on these two techniques, SISG can be fully trained on 9.5 trillion training samples in approximately 13 hours with 32 workers, which makes it possible for daily update at Taobao.
- **Practicability**: Because SISG is applied directly to recording user click sessions, the data preprocessing and preparation pipeline is straightforward and simple. Moreover, after a simple enrichment of user click sessions with SI instances, the resulting training data may be fed directly into any standard SGNS implementation, such as word2vec [14], [15]. Through this paper, we introduce in detail hands-on experiences in training and deploying billion-scale embedding models at Taobao, and we believe it can benefit practitioners working in this area.

The remainder of this paper is organized as follows: In Section II we describe the fundamentals of the SISG framework. We then present our implementation of a distributed, highly scalable training algorithm for SISG in Section III. Furthermore, in Section IV we present both offline and online experimental results and scalability performance of SISG framework. Finally, we review related work and conclude this work in Sections V and VI, respectively.

II. Method

As introduced in Section I, we are working on the matching stage, where the key task is the item-to-item similarity computation. When a user interacts with an item, e.g. clicks on an item or adds it to the shopping cart, a candidate set of items to be displayed can be retrieved according to their similarity with the previously interacted item. Recording all user click sessions over a period of several days, we obtain a set $\mathcal{S}$ of user behavior sequences, where each entry $S_u = (v_1, v_2, \cdots v_p) \in \mathcal{S}$ is a specific sequential behavior of user u in one session,[1] as shown in Figure 1(a).

[1]The duration of each session depends on different scenarios implemented in our log parsers and may vary between one hour and one day.

A. Brief review of SGNS

Given a user behavior sequence $S_u = (v_1, v_2, \cdots v_p) \in \mathcal{S}$, the goal of SGNS is to learn a low dimensional representation $\mathbf{v}_i \in \mathbb{R}^d$ for each item v_i, such that similar items are closer in the embedding space. Formally, the objective of this framework is to learn low-dimensional representation for items from the entire behavior set $\mathcal{S}$. Motivated by the well-known word2vec [14], [15], we need to maximize the following objective function $\mathcal{L}$ over $\mathcal{S}$:

$$\mathcal{L} = \sum_{S \in \mathcal{S}} \sum_{v_i \in S} \sum_{-m \le j \le m, j \neq 0} \log Pr(v_{i+j}|v_i). \quad (1)$$

$Pr(v_{i+j}|v_i)$ denotes the probability of observing a context item v_{i+j} given the target item v_i and is defined using the softmax function:

$$Pr(v_{i+j}|v_i) = \frac{\exp(\mathbf{v}_i^T \mathbf{v}'_{i+j})}{\sum_{l=1}^{|\mathcal{I}|} \exp(\mathbf{v}_i^T \mathbf{v}'_l)}, \quad (2)$$

where $\mathbf{v}_i$ and $\mathbf{v}'_i$ are the input and output embeddings of item v_i. The context is defined by a fixed-size window of length $m > 0$ centered around a target item v_i. $\mathcal{I}$ is the set of all items at Taobao. We can see that the proposed framework actually models the co-occurrence of items in user behavior sequences, so that items with similar context will be closer in the embedding space.

In practice, it is computational infeasible to optimize Eq. (1). Hence, the negative sampling method is adopted [14], [15]. In particular, a set $\mathcal{D}_p$ of positive pairs (v_i, v_j) is constructed, where v_i is the target item, and v_j is the context, which is an item occurring within a window of length m centered around item v_i. Additionally, a set $\mathcal{D}_n$ of negative pairs (v_i, v_t) is constructed, where v_i is the target item, and v_t is randomly sampled from the entire item set $\mathcal{I}$. Then, the optimization objective (1) becomes

$$\max \sum_{(v_i, v_j) \in \mathcal{D}_p} \log \sigma(\mathbf{v}_i^T \mathbf{v}'_j) + \sum_{(v_i, v_t) \in \mathcal{D}_n} \log \sigma(-\mathbf{v}_i^T \mathbf{v}'_t), \quad (3)$$

where σ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. At Taobao, the ratio of negative samples to positive ones is empirically set to 20 in production environment.

B. Side Information Enhanced Skip Gram framework (SISG)

Based on the Skip-Gram method, item embeddings can be learned and then used to compute the similarity between items. The effectiveness of this model comes from the fact that it is able to capture the co-occurrence of two items in user behavior sequences. However, for those items which occur infrequently, even in the extreme case that new items have no user interactions, it is difficult to learn accurate representations. These problems, denoted as *sparsity* and *cold start*, have been the major challenges facing RSs, and in the literature, extensive works have been explored to address these challenges by incorporating heterogeneous SI [6], [20], [28], [29]. Likewise, at Taobao, we turn to SI for these challenges.

Table I: Item and User Features used for SISG. All features take discrete integer values. In the final training sequences they are encoded as `[FeatureName]_[FeatureValue]`, e.g. "leaf_category_1234".

Item	top_level_category, leaf_category, shop, city, brand, style, material, age_gender_purchase_level (cross feature)
User	age_gender (cross feature), user_tags

In this paper, we refer to as SI metadata of items or users, e.g. the category of an item, or the gender of users. It is intuitive to assume that items with similar SI should also be similar and thus closer in the embedding space. To make use of these SI at Taobao, we design a flexible mechanism as follows: given a user behavior sequence $S_u = (v_1, \cdots, v_p)$ for user u we enrich this sequence by injecting both item SI SI_i^k "nodes" and "user types" UT_u. The enriched sequence, denoted as $\tilde{S}$, then takes the form

$$v_1, SI_1^1, \cdots, SI_n^1, \cdots, v_p, SI_1^p, \cdots, SI_n^p, UT_u, \quad (4)$$

where SI_k^i denotes the k-th SI for item v_i, UT_u represents the user types for user u, n is the number of all SI. In this context, a user type is understood as a fine grained categorization of users based on a combination of various user metadata, e.g. one particular user type could contain all female users aged between 31 and 35, who are married, have children and posses a car. Similar to all other possible SI, every user type UT_u is represented in the form

`UT_[gender]_[age]_[t1]_[t2]_[t3]_...`

where for instance `gender` could be "F" (for female), `age` could be "19-25", and all tags are substituted with indicators `t1,t2,t3,`$\cdots$, which further characterize a user type. For the above example, we would for instance obtain `married_haschildren_hascar`. The number of tags per user type may vary. See Table I for an overview of the item and user meta data used in our experiments, which have been demonstrated useful for recommendation at Taobao.

The entire "SI-enhanced" sequences of the form (4) can then be fed into any standard SGNS implementation, which will output the final embeddings $\mathbf{v}_j$, $\mathbf{SI}_k^i$, and $\mathbf{UT}_u$. The framework of SISG is shown in Figure 1(c). It is clear that by construction this will ensure that the embeddings of items which have often been clicked within one session, items with similar SIs, items clicked on by similar user types, and user types of users which have clicked similar items, will all lie to close to each other in one semantic vector space. Here "closeness" is understood as a large cosine similarity between two vectors.

C. Capturing Asymmetric User Behavior

In the standard skip-gram method, as shown in Section II-A, when we construct positive samples from a given user behavior sequence, (v_i, v_j) are sampled from a symmetric window $W_m(v_i) = \{v_{i+j} \mid -m \le j \le m, j \neq 0\}$ around the center item v_i. However, it ignores the order information in the user behavior sequence. For example, given a user behavior sequence $S = (v_1, v_2, \cdots, v_p)$ for user u, without loss of generality, assuming $i < j$, we can construct two

positive samples (v_i, v_j) and (v_j, v_i) when using v_i and v_j, respectively, as target items. However, from the sequence, we know that user u clicks v_j after clicking v_i, but not the other way around. In other words, we should recommend v_j as candidate set of similar items for v_i, but not vice versa. We refer to this phenomenon as "asymmetry" underlying user behaviors, and it is clearly problematic to ignore this asymmetry. Hence, in order to capture asymmetry of user behavior, it is reasonable to restrict sampling to only the right or left context window of an item v_i (depending on how the sequence is ordered). Since items in our sequences are ordered from left to right w.r.t the sequence of actions of a user, we thus sample skip-grams only from the right context window of very element in a sequence.

In the Skip-Gram framework, for an item v_i, we obtain two groups of vectors: an input vector $\mathbf{v}_i$ when the item is used as target item, and an output vector $\mathbf{v}'_i$, when it is used as context item. In most previous work using Skip-Gram embeddings for similarity calculation output vectors $\mathbf{v}'_i$ are discarded and similarities are obtained from the inner product between input vectors only. While this can be justified, as long as (v_i, v_j) are sampled from symmetric context windows, this is no longer reasonable when using only the left or right context window. We thus proceed along the lines of [18] and [31], where similar methods are discussed in the context of preserving "asymmetric transitivity" for graph embedding. In particular, we also compute similarities for pairs (v_i, v_j) and (v_j, v_i) via $\mathbf{v}_i^T \mathbf{v}'_j$ and $\mathbf{v}_j^T \mathbf{v}'_i$, respectively.

In our experiments, we have been able to observe a significant increase in recommending accuracy, once we take into account different directions in our SISG implementation. This is in line with what we typically observe in our user behavior sequences. On average, for our users we estimate the probability for a significant difference between the number of clicks $v_i \rightarrow v_j$ and $v_j \rightarrow v_i$ to be around 20%. In Section IV-A, the experimental results demonstrate the effectiveness of incorporating asymmetry of user behaviors.

D. Comparison with our Previous Work

Previously, our team have designed a graph embedding based framework for recommendation [23], where an item graph is firstly constructed from user behavior sequences and then item sequences are generated using a random walk on the constructed graph. These sequences are then fed into a modified SGNS framework. To alleviate the cold start problem, item meta data are incorporated in a manner shown in Figure 1(b). Despite its superiority in capturing higher-order similarity between items over conventional CF-based methods, here we point out several limitations of EGES, which motivate us to design the proposed SISG framework.

A major problem of EGES is the information loss that occurs when the item graph is constructed from user behavior sequences. Specifically, we lose the link between user IDs and the behavior sequences. Thus, user metadata cannot be employed, especially for cold start users, who have no records of previous purchases or browsing sessions. Further, as mentioned in Section I, it also ignores the asymmetry of user behaviors, leading to a suboptimal solution. Lastly, when deploying EGES in our production environment, the item graph is split along item categories into a number of smaller subgraphs, each of which has less than 50 million nodes. Thus, each subgraph can be processed by one worker and trained in parallel. However, edges between subgraphs are removed, which leads to information loss and subgraphs are embedded in different semantic spaces. Hence, despite EGES's performance gain compared to well-tuned CF [23], the proposed SISG framework in this work have evident advantages.

III. Distributed Training Mechanism

Clearly, the SISG framework could be implemented using any standard implementation such as word2vec. [2] However, this is challenging when the "vocabulary" size, i.e. the number of items and SI is in the order of billions (we obtain around 5×10^{10} "tokens" when training SISG for online A/B test). Thus, a distributed training mechanism has to be used.

Due to the structure exhibited by typical SGD-based optimization algorithms for learning skip-gram models, a distributed training algorithm becomes very difficult to implement, once all input and output vectors do not fit into a single machine's memory. To address this problem we have deployed a variant of the TNS-algorithm (Target Negative Sampling) [21]. The general structure of this algorithm is given in Algorithm 1.

The basic idea is as follows: First, all items v_i are assigned to one partition per worker. Every worker constructs its own local noise distribution and then independently samples pairs (v_i, v_j) from the entire behavior sequences. If v_i is not managed by Worker A, the pair is ignored. Otherwise the output vector for v_i is sent to the Worker A' , which manages v_j. On A' negative samples are drawn from the local noise distribution. All gradients for input and output vectors, as well as gradient updates for output vectors are computed. Eventually the gradient w.r.t the input vector of v_i is returned to Worker A, which performs the final gradient update.

For any distributed Skip-Gram implementation, once vectors are distributed across several machines each managing only a partition of all vectors, very high communication costs may arise, especially considering that negative samples are drawn from a global noise distribution when following standard Skip-Gram recipes. In general, following the TNS approach, communication costs are already kept somewhat in check. Nevertheless, for our particular case this has not been proven optimal. We have thus employed an adapted Target Negative Sampling Algorithm, which is characterized by a smart partitioning strategy based on HBGP and a special treatment of vectors for high frequency SI.

A. Adapted Target Negative Sampling

At Taobao, we still face another challenging problem when applying the TNS approach, which is the extremely imbal-

[2] https://code.google.com/archive/p/word2vec/

Algorithm 1 Distributed SISG using TNS.

for $S \in \mathcal{S}$ **do**
  for $v_i \in S$ **do**
    Sample pair (v_i, v_j) from S;
    A := Current worker
    A' := Worker processing v_j
    if v_i processed by A and v_j processed by A' **then**
      Get $\partial_{\mathbf{v}_i} L$ by calling TNS($\mathbf{v}_i$, v_j) on A';
      $\mathbf{v}_i := \mathbf{v}_i - \eta \partial_{\mathbf{v}_i} L$
    end if
  end for
end for
function TNS($\mathbf{v}_i$, v_j)
  Compute $\partial_{\mathbf{v}'_j} L$.
  $\mathbf{v}'_j := \mathbf{v}'_j - \eta \partial_{\mathbf{v}'_j} L$
  Sample N negatives v_t from local noise distribution;
  for all negative samples v_t **do**
    $\mathbf{v}'_t := \mathbf{v}'_t - \eta \partial_{\mathbf{v}'_t} L$
  end for
  Compute $\partial_{\mathbf{v}_i} L$
  return $\partial_{\mathbf{v}_i} L$
end function

anced distribution of "word frequencies" for items and SI. Hot items tend to occur in most user behavior sequences, and some types of SI columns might contain only a small number of distinct feature values (e.g. "Gender" takes only three values: "female", "male", "null"). Thus, a very large number of pairs may end up being processed by a single worker, while the remaining workers have already finished processing. As this will impact both the overall efficiency, and lead to convergence problems, we have modified TNS in the following way:

- High frequency "words" are aggressively down sampled.
- Our implementation of TNS allows the top-K frequent items to be kept in all partitions at the same time. The corresponding vectors are then synchronized (averaged) at regular intervals.

We denote this approach as Adapted TNS (ATNS).

Since most high frequency "words" are SIs, the above two points do not address the problem of hot items. Also, they do not take into account that most sequences contain items from one leaf category only, which, when properly exploited, could decrease communication costs even further. To leverage this structure, we have further implemented a smart partitioning strategy, which will be presented in the next section.

B. Heuristic Balanced Graph Partitioning

As outlined in the previous section, it is still possible to reduce communication costs further. A key observation regarding our user behaviors makes this possible. That is that most Taobao users tend to view items from one leaf category only within one browsing session. Thus we can split the data along item leaf categories, which implies that for a sampled pair (v_i, v_j) the probability of both items being managed by the same worker increases. This obviously lowers the number of remote calls of the TNS function in Algorithm 1 and thus decreases communication costs.

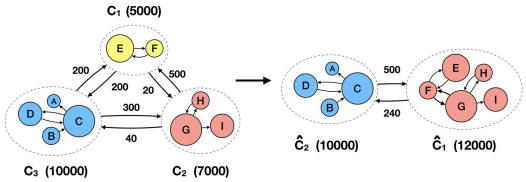


Fig. 2. HBGP strategy. We illustrate our partition strategy with this example. In the current step, we have grouped all the leaf categories into three groups, denoted as C_1, C_2, C_3, and the corresponding figures are the total frequency of items in each group. The weights of edges connecting two groups are the total transition frequency between items across these two groups. According to the heuristic strategy, it is better to merge groups C_1 and C_2 into a new group $\hat{C}_1$ and keep C_3, denoted as $\hat{C}_2$, after considering the communication (transition frequency should be as small as possible between groups) and computational (item frequency should be closer across different groups) costs.

However, the number of workers is typically much smaller than that of leaf categories on Taobao. Thus, we have to put the items of several leaf categories in one partition. Our goal is to design this assignment in such a way that

1) The overall frequency of all items in each worker should be about the same.
2) The probability of the two items in a pair to be assigned to two different workers should be small.

To achieve a balanced assignment given the number of partitions w, i.e., number of workers, we design the following HBGP strategy (an illustrative example is given in Figure 2):

1) Construct a directed weighted item graph $\mathcal{G}$ from user behaviors sequences $\mathcal{S}$, where the weight of each edge is the total transition frequency of two nodes in all behavior sequences.
2) Reduce the item graph G to a graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ containing only leaf-category nodes. The transition frequency between two leaf-category nodes in $\tilde{\mathcal{G}}$ is the sum of frequencies of edges in $\mathcal{G}$ that connect the same two leaf categories. Moreover, for $C \in \tilde{\mathcal{V}}$, let $|C|$ denote the number of times the items from category C appear in the set of training sequences.
3) Iteratively merge nodes of $\tilde{\mathcal{G}}$:
   a) Find the edge (C_1, C_2) with the largest sum of transition frequencies for the two directions.
   b) Merge C_1 and C_2, if $|C_1|+|C_2| \leq \beta \cdot |\mathcal{V}|/w$, where $\beta \geq 1$ is a user defined parameter specifying the maximally imbalance allowed. Otherwise goto (a).
   c) Recalculate all transition frequencies accordingly after merging C_1 and C_2.
   d) Stop when $\tilde{\mathcal{G}}$ only consists of w nodes, i.e., the required number of partitions is obtained.
   e) If $|C_1| + |C_2| \leq \beta \cdot |\mathcal{V}|/w$ holds for none of the edges, increase β and repeat (3).

Using this heuristic strategy, we succeed in training our SISG framework with 800 million items, while still allowing for a further scale up to 2 billion or more items by increasing the number of workers. We present the scalability performance in Section IV-D. In our production environment, β is set

to 1.2 empirically. According to the existing literature, this strategy is related to a variant of the well-known K-minimum cut problem [5], where the output partitions have pre-defined sizes. Better approximation algorithms in our scenarios are left for future work. We also mention that the above method only assigns items to partitions, but not SI or user types. For these we rely on the caching (averaging) strategy outlined in the previous section, as long as their frequencies are large enough to impact the overall computational efficiency. We now conclude this section with an overall description of the entire training process.

C. Training Pipeline

Overall, our training process consists of the following stages:

1) Transform all item sequences $\mathcal{S}$ into the enriched sequences $\tilde{\mathcal{S}}$ according to (4).
2) Count item, user type and SI frequencies in $\tilde{\mathcal{S}}$ and store them in a dictionary $\mathcal{D}$,
3) Partition $\mathcal{D}$ into $(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_w)$, where $w > 0$ denotes the number of workers. The target partition for items is determined according to the strategy outlined in Section III-B, while the target partitions for SI and user types are assigned randomly.
4) Determine the set of shared nodes $\mathcal{Q}$ containing all elements in $\mathcal{D}$ with frequency above a certain threshold. In our setting, this has the effect of $\mathcal{Q}$ usually containing the most common SI features such as age, gender, color, etc.

The training process is performed according to Algorithm 1.

We mention further the implementation details regarding pair sampling and generation of negative samples:

- We have implemented the sampling of pairs (v_i, v_j) following the standard word2vec implementation, with an additional option to switch to using the right context window only in order to train directional models. Since all of our training sequences have a fixed maximal length, we can adjust the window size, such that all possible pairs per sequence are sampled.
- We also apply the same method of subsampling the very frequent pairs as proposed originally in [16]. In our setting it has proven useful to agressively downsample very frequent pairs caused by some of the additional SI.
- Every worker maintains its own noise distribution for the elements of $\mathcal{P}_j \cup \mathcal{Q}$, where the probability of drawing v as a negative sample is modeled in such a way that

$$P_{\text{noise}}(v) \sim \text{freq}(v)^{\alpha},$$

where we use the standard choice of $\alpha = 0.75$.

IV. Experiments

In this section, we present extensive experimental results to demonstrate the effectiveness and scalability of SISG for recommendation at Taobao.

Table II: Statistics of three datasets. Taobao25M is used for offline experiments, and Taobao100M is used for online A/B test. We further train SISG on even larger datasets, such as Taobao800M, which consists of 800 million items. In practice, we train and deploy SISG online on the two larger datasets. Since they exhibited similar CTR gains, we use the smaller Taobao100M dataset to support daily recommendation for the Taobao Homepage scenario and train on the Taobao800M dataset for other downstream tasks which need full data. Here "#tokens" represents the total number of items and their corresponding SI instances.

	Taobao25M	Taobao100M	Taobao800M
#Items	25,549,673	105,412,789	806,166,969
#SI	8	8	8
#User types	214,097	228,958	250,409
#Tokens	$\sim 2.3 \times 10^{10}$	$\sim 5.0 \times 10^{10}$	$\sim 2.0 \times 10^{11}$
#Positive pairs	$\sim 2.0 \times 10^{11}$	$\sim 4.6 \times 10^{11}$	$\sim 1.8 \times 10^{12}$
#Training pairs	$\sim 1.2 \times 10^{12}$	$\sim 9.5 \times 10^{12}$	$\sim 3.7 \times 10^{13}$

A. Offline Evaluation

Datasets. To conduct offline experiments, we create a dataset containing user behavior sequences collected over seven days. Denoted as Taobao25M, it contains roughly twenty five million Taobao items. The corresponding SIs are also collected. The statistics of Taobao25M is shown in Table II.

Next Item Recommendation. To evaluate our framework, we choose the next item recommendation task, which aims at recommending an item to a user given his/her behavior sequence. In particular, behavior sequences of all users are taken. For each sequence $S = (v_1, v_2, \cdots, v_p)$, we first use $(v_1, v_2, \cdots, v_{p-2})$ for training, and tune SISG based on the performance on v_{p-1} to obtain the best hyper-parameters. Then we train SISG on $(v_1, v_2, \cdots, v_{p-1})$ and report the performance of SISG on v_p. After training SISG, we use the embeddings to retrieve the K most similar items, denoted as $S_K(v_{p-1})$ to item v_{p-1}. To evaluate the performance, we use HitRate (HR) as our metric, which is defined as:

$$\text{HR@K} = \frac{1}{|\mathcal{S}|} \sum_{S \in \mathcal{S}} \mathbb{I}(v_p \in S_K(v_{p-1})), \tag{5}$$

where $\mathbb{I}(\cdot)$ is the indicator function. $\mathcal{S}$ is the set of all user behavior sequences.

Variants of SISG. To show the influence of different components of SISG, we use the following model variants:

- **SGNS**: Classic SGNS applied to sequences of items only (no SI, no user types).
- **EGES**: Our previous work on top of the graph embedding framework [23], which only incorporates item metadata as SI.
- **SISG-F**: SISG with sequences of the form (4) without addition of user types. Asymmetry of user behaviors are ignored.
- **SISG-U**: SISG applied to sequences of items only (no SI) with the addition of user-types.
- **SISG-F-U**: SISG-F with additional injection of user types into the sequences.
- **SISG-F-U-D**: SISG-F-U extended to account for asymmetry of user behaviors.

In all experiments, we compute similarities using the standard cosine similarity. We choose $d = 128$ as the embedding

Table III: HRs of different variants of SISG. The performance gain comparing to SGNS is shown next to the corresponding metric. The best performance gain is highlighted in bold, which is obtained by SISG-F-D-U, demonstrating the effectiveness of every component in SISG: heterogeneous SI, including metadata of items and users, as well as asymmetry in user behaviors.

Variants	**HR@1**	increase	**HR@10**	increase	**HR@20**	increase	**HR@100**	increase	**HR@200**	increase
SGNS	0.0043	-	0.0119	-	0.0150	-	0.0248	-	0.0308	-
EGES	0.0041	-5.65%	0.0143	20.17%	0.0186	24.00%	0.0321	29.44%	0.0400	29.87%
SISG-F	0.0054	25.58%	0.0174	46.22%	0.0224	49.33%	0.0370	49.19%	0.0450	46.10%
SISG-U	0.0051	18.60%	0.0145	21.85%	0.0180	20.00%	0.0289	16.53%	0.0353	14.61%
SISG-F-U	0.0057	32.56%	0.0184	54.62%	0.0238	58.67%	0.0391	57.66%	0.0474	53.90%
SISG-F-U-D	**0.0078**	**81.40%**	**0.0293**	**146.22%**	**0.0395**	**163.33%**	**0.0702**	**183.06%**	**0.0863**	**180.19%**

space dimension and train for $T = 2$ epochs using $N_{\text{neg}} = 20$ negative samples. The results of our offline evaluation are presented in Table III. From these results we have the following observations:

- SISG-F-U-D outperforms all the other variants in HRs with different Ks, which demonstrates the effectiveness of every component in SISG, i.e., heterogeneous SI, including metadata of items and users, and capturing the asymmetry of user behaviors. In particularly, compared to basic SGNS, the performance gain is 180.19% in HR@200, which is very significant in practice.
- When comparing SISG-F and EGES with SGNS, we can see that the performance gain of SISG-F is larger than that of EGES. Note that both SISG-F and EGES are using the same SI. Thus, through the novel design in Section II-B, SISG can make more effective use of SI than EGES. Another way to explain the superior performance of SISG-F over EGES is from the perspective of combination of positive pairs between item and SIs. As shown in Figures 1(b) and 1(c), the positive pairs for EGES only include the combinations like "input item, output item" and "SI of input item, output item", while those for SISG-F include more combinations like "input item, SI of output item" or "SI of input item, SI of output item". In this sense, we can thus argue that the SISG-F model is more expressive than EGES. In terms of model parameters, this is reflected by the fact that in the EGES model SI vectors do not have corresponding output vectors, while in the SISG-F model every SI possesses both an input and an output vector.
- When comparing SISG-U, SISG-F with SGNS, we can see that the performance gain by SISG-F is much higher than that of SISG-U. This implies that item SI plays a more important role than user types.

B. Online Evaluation

In this section, we report the online performance of SISG-F-U-D deployed in our production systems in an A/B test setting. Our metric is Click-Through-Rate (CTR) on the homepage of the Mobile Taobao App. We run SISG on around 100 million items, denoted by Taobao100M as shown in Table II. We use SISG-F-U-D to generate a number of similar items as recommendation candidates for each item. The final recommendation results on the homepage in Taobao are generated by the ranking engine, which is implemented based on a deep neural network model. We use the same method to rank the candidate items in the experiment. We report the performance comparisons between SISG-F-U-D with well-tuned CF during an 8-day period in January 2019. The results are shown in Figure 3.

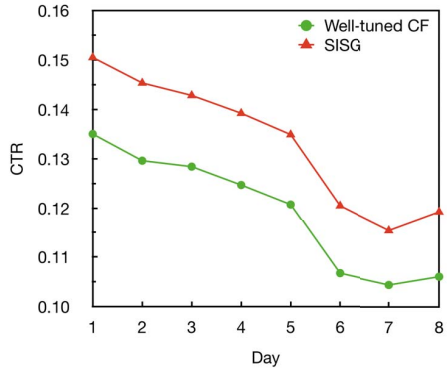


Fig. 3. Online CTRs of different methods during eight days in January 2019.

From Figure 3, we can see that SISG-F-U-D outperforms well-tuned CF very significantly (the improvement is 10.01%), which demonstrates the effectiveness of SISG. Note that we did not deploy EGES online as it is much more time-consuming than CF to deploy in the production systems at Taobao. Thus, it is not considered due to the inferior performance to SISG in production environment for the sake of computational resource. However, from the experimental results reported in last year [23], the performance gain by EGES over the same well-tuned CF is around 3% during a 7-day period around the Double Eleven Festival in 2017. Taking into consideration this and the offline experimental results in Section IV-A, we can see that SISG outperforms EGES in a large margin.

C. Case Study

In this part, we present a few real-world cases to show some interesting observations based on the embeddings learned by SISG from billion-scale sets of Taobao users and items.

1) Cold Start Users: At Taobao, it remains challenging to recommend for cold start users, i.e. users who have no behavior histories. In that case, we may take the average of all user type vectors matching a particular user. For example, if we would like to recommend items to a female user between

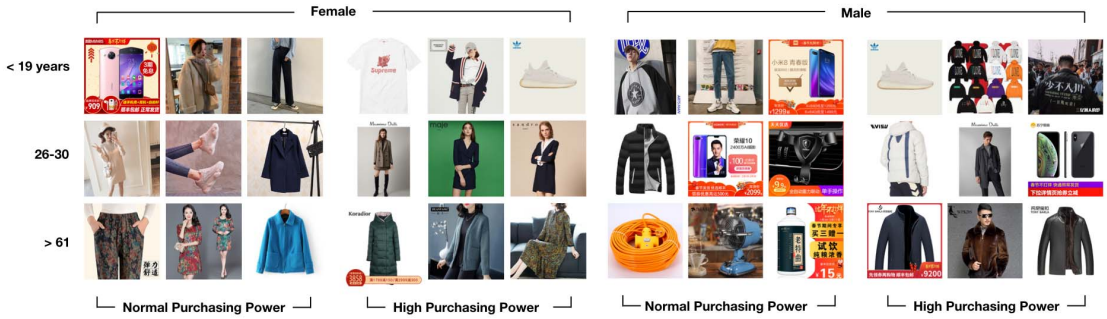


Fig. 4. Example: Cold start recommendations for different user groups. We compare cold start recommendations for different user groups and observe clear differences in user behavior regarding gender, age, and purchasing power. While the differences between female and male users are obvious, we also note that both female and male users with higher purchasing power are recommended more items from expensive brands (Adidas shoes, Apple iPhone). The differences between age groups are especially pronounced for male users, where recommended items for relatively young users and seniors differ greatly.

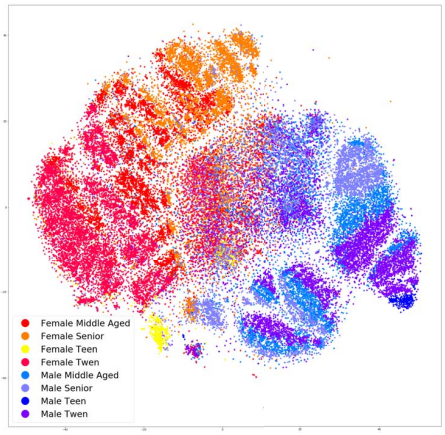


Fig. 5. Example: t-SNE [13] plot of user type embeddings. Evidently, users show different behaviors in terms of gender and age.

21 and 25 years old, we can take the average of all user type vectors which belong to a user type containing the "female" and "age 21-25" features. Example recommendations using this approach are shown in Figure 4, which contains recommendations generated for different groups of users. We can observe the distinctive difference across different groups of users. For example, fashionable clothes or shoes tend to be recommended to female users between 26 and 30 years old, while iPhones tend to be recommended to male users with higher purchasing power between 25 and 30.

From Figure 4, it is clear that female and male users have significantly different recommendation results, which are consistent with general trends among Taobao users. We further show a t-SNE [13] plot of roughly 50,000 user type vectors in Figure 5. As is clearly visible, "male" and "female" user type vectors concentrate in different regions of the embedding space, and within each region, clusters corresponding to different age groups are visible.

2) Cold Start Items: We briefly sketch how the SISG framework supports recommendation of cold start items, for which there is no training data available. Suppose that we are given a new item v, for which there exists no user interaction data relating it to other items. Here we "infer" an embedding vector v for v_j via

$$\mathbf{v} = \sum_{k=1}^{s} \mathbf{SI}_k(v), \tag{6}$$

with $\mathbf{SI}_k(v)$ denoting the SI vectors corresponding to the metadata of the new item v. We then proceed by finding the most similar vectors for $\mathbf{v}$. In Figure 6 we give an example for such cold-start recommendations obtained via (6).

Fig. 6. Example for cold start item recommendation: For the item on the left we obtain recommendations by using the trained item vector (top right row) and compared it to recommendations obtained using (6), which uses SI vectors only (bottom right row).

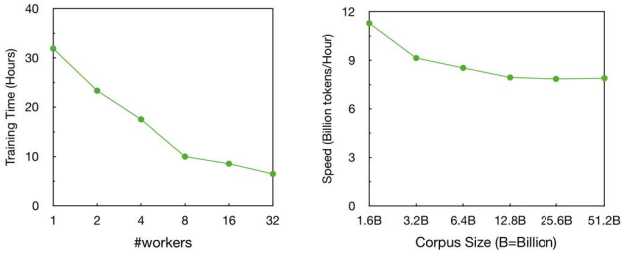
D. Scalability

As introduced in Section III, to implement SISG at Taobao where billions of items are available, we implemented a distributed mechanism to train SISG with two key practical components: ATNS and HBGP. In this section, we conduct two experiments to demonstrate the scalability of SISG.

First, we train SISG with different number of workers and expect the training time to be proportional to the inverse of the number of workers, as explained in Section III. To show this, we train SISG on Taobao100M dataset. The statistics of this dataset are shown in Table II. All the tests are run on a cluster of machines with 480G memory, 50 CPU cores at 2.5GHZ, and 10Gbps Ethernet. Training time are reported in Figure 7(a), which show that with the number of workers increasing, the training time decreases. In particular, the trend is very close to the function $y = \frac{1}{x}$, which demonstrates that the training time is proportional to the inverse of the number of workers.

In the second experiment we measure training time while varying the corpus size, i.e. the total number of all tokens including items and SI. The experiments are conducted on a fixed number of workers (32). We use "billion tokens per hour" as the definition of training speed to evaluate the performance. The experimental results are shown in Figure 7(b). We can see that the speed decreases when the corpus size increases, and becomes relatively stable when the corpus size is beyond 12.8 billion.

From the above two results, we can see that SISG is highly-scalable and able to deal with billions of real-world items at Taobao.



(a) Training time w.r.t different number of workers.

(b) Training speed w.r.t. different corpus sizes, i.e., number of tokens.

Fig. 7. Training time of SISG on Taobao100M dataset.

V. Related Work

In this section, we review the related work, including skip-gram models for recommendation and large-scale distributed skip-gram.

A. Skip-Gram Models for Recommendation

Skip-gram models were first used in the context of word embeddings (e.g. see the seminal work of Mikolov [14], [15]) and later for graph embedding problems [8], [19] by learning an embedding from a "corpus" of random walk or similarly generated node sequences. Given the success of word2vec and graph embedding based methods, skip-gram models have been applied to recommendation scenarios to learn embeddings of items, which are used to compute similarities. This approach has been explored previously in [6], [7], [22]–[24], [26], [28]–[31]. In this work, by revisiting these works, we design a flexible skip-gram framework at Taobao, which integrates the advantages of previous works. Moreover, we deploy our framework in real-world production environment for a billion-scale set of users and items.

B. Large-Scale Distributed Skip-Gram

Initially, distributed implementations for Skip-Gram with Negative Sampling adopted a pure data parallelism strategy [1], [9]. In order to allow larger vocabularies, Ordentlich et al. [17] proposed sharding of embedding vectors by dimension, rather than than nodes. For our work we have extended the TNS method first presented in [21], which was studied in the context of word embeddings for text corpora with very large vocabularies. Besides, we further design a HBGP strategy, contributing to successful training of SISG on a billion-scale set of items at Taobao.

VI. Conclusion and Future work

Motivated by the success of embedding methods, we present a flexible and highly scalable skip-gram model, SISG, to address challenges of embedding models for recommendation at Taobao. SISG leverages heterogeneous SI, including item and user metadata and captures the asymmetry of user behaviors to achieve significantly superior recommendation performance over EGES and other baselines. In addition, to be able to train SISG on a billion-scale set of items and users, we design an effective distributed mechanism featuring an adaptive target negative sampling (ATNS) technique and heuristic balanced graph partition strategy (HBGP). By conducting extensive experiments, including offline and online evaluations, real-world case studies, and scalability tests, we demonstrate the effectiveness of SISG in real-world recommendation scenarios at Taobao. In summary, this work presents an effective and efficient embedding-based framework for practitioners working on similar large-scale recommendation applications.

For future work, we plan to include other types of SI into SISG, in particular, textual and visual information at Taobao.

VII. Acknowledgments

Huan Zhao and Dik Lun Lee were supported by the Research Grants Council HKSAR GRF (No.16215019). We would like to thank colleagues of our team - Wei Li, Qiwei Chen, Chao Li, Zhiyuan Liu, Yuchi Xu, Mengmeng Wu, Jiaming Xu and Wen Chen for useful discussions and supports on this work. We are grateful to Zilin Lee, Qiyi Huang and Zhengping Qian from Alibaba graph computation team with whom we cooperated for this work. We also thank the anonymous reviewers for their valuable comments and suggestions that help improve the quality of this manuscript.

References

[1] Apache spark, mllib - word2vec implementation. https://spark.apache.org/docs/latest/mllib-feature-extraction.html\#word2vec.

[2] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. Behavior sequence transformer for e-commerce recommendation in alibaba. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, pages 1–4, 2019.

[3] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide and deep learning for recommender systems, 2016.

[4] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *RecSys*, pages 191–198, 2016.

[5] Olivier Goldschmidt and Dorit S Hochbaum. Polynomial algorithm for the k-cut problem. In *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pages 444–451, 1988.

[6] Mihajlo Grbovic and Haibin Cheng. Real-time personalization using embeddings for search ranking at airbnb. In *SIGKDD*, pages 311–320, 2018.

[7] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. E-commerce in your inbox: Product recommendations at scale. In *SIGKDD*, pages 1809–1818, 2015.

[8] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864, 2016.

[9] Shihao Ji, Nadathur Satish, Sheng Li, and Pradeep Dubey. Parallelizing word2vec in shared and distributed memory. *arXiv preprint arXiv:1604.04661*, 2016.

[10] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *SIGKDD*, pages 426–434, 2008.

[11] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Huan Zhao, Pipei Huang, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. Multi-interest network with dynamic routing for recommendation at tmall. In *CIKM*, pages 2615–2623, 2019.

[12] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, (1):76–80, 2003.

[13] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research (JMLR)*, 9(Nov):2579–2605, 2008.

[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[15] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NeurIPS*, pages 3111–3119, 2013.

[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NeurIPS*, pages 3111–3119, 2013.

[17] Erik Ordentlich, Lee Yang, Andy Feng, Peter Cnudde, Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, and Gavin Owens. Network-efficient distributed word2vec training system for large vocabularies. In *CIKM*, pages 1139–1148, 2016.

[18] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *SIGKDD*, pages 1105–1114, 2016.

[19] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *SIGKDD*, pages 701–710, 2014.

[20] Chuan Shi, Binbin Hu, Xin Zhao, and Philip Yu. Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2018.

[21] Stergios Stergiou, Zygimantas Straznickas, Rolina Wu, and Kostas Tsioutsiouliklis. Distributed negative sampling for word embeddings. In *AAAI*, pages 2569–2575, 2017.

[22] Flavian Vasile, Elena Smirnova, and Alexis Conneau. Meta-prod2vec: Product embeddings using side-information for recommendation. In *RecSys*, pages 225–232, 2016.

[23] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *SIGKDD*, pages 839–848, 2018.

[24] Ledell Yu Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, and Jason Weston. Starspace: Embed all the things! In *AAAI*, pages 5569–5577, 2018.

[25] Wenyi Xiao, Huan Zhao, Haojie Pan, Yangqiu Song, Vincent W. Zheng, and Qiang Yang. Beyond personalization: Social content recommendation for creator equality and consumer satisfaction. In *SIGKDD*, pages 235–245, 2019.

[26] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*, pages 974–983, 2018.

[27] Huan Zhao, Quanming Yao, James T Kwok, and Dik Lun Lee. Collaborative filtering with social local models. In *ICDM*, pages 645–654. IEEE, 2017.

[28] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. Meta-graph based recommendation fusion over heterogeneous information networks. In *SIGKDD*, pages 635–644, 2017.

[29] Huan Zhao, Yingqi Zhou, Yangqiu Song, and Dik Lun Lee. Motif enhanced recommendation over heterogeneous information network. In *CIKM*, pages 2189–2192, 2019.

[30] Kui Zhao, Yuechuan Li, Zhaoqian Shuai, and Cheng Yang. Learning and transferring ids representation in e-commerce. In *SIGKDD*, pages 1031–1039, 2018.

[31] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. Scalable graph embedding for asymmetric proximity. In *AAAI*, pages 2942–2948, 2017.