Learning Tree-based Deep Model for Recommender Systems

Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, Kun Gai
Alibaba Group
{zhuhan.zh,yushi.lx,pengye.zpy,guozheng.lgz,jay.hj,lihan.lh,jingshi.gk}@alibaba-inc.com

ABSTRACT

Model-based methods for recommender systems have been studied extensively in recent years. In systems with large corpus, however, the calculation cost for the learnt model to predict all user-item preferences is tremendous, which makes full corpus retrieval extremely difficult. To overcome the calculation barriers, models such as matrix factorization resort to inner product form (i.e., model user-item preference as the inner product of user, item latent factors) and indexes to facilitate efficient approximate k-nearest neighbor searches. However, it still remains challenging to incorporate more expressive interaction forms between user and item features, e.g., interactions through deep neural networks, because of the calculation cost.

In this paper, we focus on the problem of introducing arbitrary advanced models to recommender systems with large corpus. We propose a novel tree-based method which can provide logarithmic complexity w.r.t. corpus size even with more expressive models such as deep neural networks. Our main idea is to predict user interests from coarse to fine by traversing tree nodes in a top-down fashion and making decisions for each user-node pair. We also show that the tree structure can be jointly learnt towards better compatibility with users' interest distribution and hence facilitate both training and prediction. Experimental evaluations with two large-scale real-world datasets show that the proposed method significantly outperforms traditional methods. Online A/B test results in Taobao display advertising platform also demonstrate the effectiveness of the proposed method in production environments.

CCS CONCEPTS

• **Computing methodologies** → **Classification and regression trees**; **Neural networks**; • **Information systems** → **Recommender systems**;

KEYWORDS

Tree-based Learning, Recommender Systems, Implicit Feedback

1 INTRODUCTION

Recommendation has been widely used by various kinds of content providers. Personalized recommendation method, based on the intuition that users' interests can be inferred from their historical behaviors or other users with similar preference, has been proven to be effective in YouTube [7] and Amazon [22].

Designing such a recommendation model to predict the best candidate set from the entire corpus for each user has many challenges. In systems with enormous corpus, some well-performed recommendation algorithms may fail to predict from the entire corpus. The linear prediction complexity w.r.t. the corpus size is unacceptable. Deploying such large-scale recommender system requires the amount of calculation to predict for each single user be limited. And besides preciseness, the novelty of recommended items should also be responsible for user experience. Results that only contain homogeneous items with user's historical behaviors are not expected.

To reduce the amount of calculation and handle enormous corpus, memory-based collaborative filtering methods are widely deployed in industry [22]. As a representative method in collaborative filtering family, item-based collaborative filtering [31] can recommend from very large corpus with relatively much fewer computations, depending on the pre-calculated similarity between item pairs and using user's historical behaviors as triggers to recall those most similar items. However, there exists restriction on the scope of candidate set, i.e., not all items but only items similar to the triggers can be ultimately recommended. This intuition prevents the recommender system from jumping out of historical behavior to explore potential user interests, which limits the accuracy of recalled results. And in practice the recommendation novelty is also criticized. Another way to reduce calculation is making coarse-grained recommendation. For example, the system recommends a small number of item categories for users and picks out all corresponding items, with a following ranking stage. However, for large corpus, the calculation problem is still not solved. If the category number is large, the category recommendation itself also meets the calculation barrier. If not, some categories will inevitably include too many items, making the following ranking calculation impracticable. Besides, the used categories are usually not designed for recommendation problem, which can seriously harm the recommendation accuracy.

In the literatures of recommender systems, model-based methods are an active topic. Models such as matrix factorization (MF) [19, 30] try to decompose pairwise user-item preferences (e.g., ratings) into user and item factors, then recommend to each user its most preferred items. Factorization machine (FM) [28] further proposes a unified model that can mimic different factorization models with any kind of input data. In some real-world scenarios that have no explicit preference but only implicit user feedback (e.g., user behaviors like clicks or purchases), Bayesian personalized ranking [29] gives a solution that formulates the preference in triplets with partial order, and applies it to MF models. In industry, YouTube uses deep neural network [7] to learn both user and item's embeddings, where two kinds of embeddings are generated from their corresponding features separately. In all the above kinds of methods, the preference of user-item pair can be formulated as the inner product of user and item's vector representations. The prediction stage thus is equivalent to retrieve user vector's nearest neighbors in inner product space. For vector search problem, indices like hashing or quantization [18] for approximate k-nearest neighbor (kNN) search can ensure the efficiency of retrieval.

However, the inner product interaction form between user and item's vector representations severely limits model's capability. There

exist many other kinds of more expressive interaction forms, for example, cross-product features between user's historical behaviors and candidate items are widely used in click-through rate prediction [5]. Recent work [13] proposes a neural collaborative filtering method, where a neural network instead of inner product is used to model the interaction between user and item's vector representations. The work's experimental results prove that a multi-layer feed-forward neural network performs better than the fixed inner product manner. Deep interest network [34] points out that user interests are diverse, and an attention like network structure can generate varying user vectors according to different candidate items. Beyond the above works, other methods like product neural network [27] have also proven the effectiveness of advanced neural networks. However, as these kinds of models can not be regulated to inner product form between user and item vectors to utilize efficient approximate kNN search, they can not be used to recall candidates in large-scale recommender systems. How to overcome the calculation barrier to make arbitrary advanced neural networks feasible in large-scale recommendation is a problem.

To address the challenges above, we propose a novel tree-based deep recommendation model (TDM) in this paper. Tree and tree-based methods are researched in multiclass classification problem [1–3, 6, 15, 26, 32], where tree is usually used to partition the sample or label space to reduce calculation cost. However, researchers seldom set foot in the context of recommender systems using tree structure as an index for retrieval. Actually, hierarchical structure of information ubiquitously exists in many domains. For example, in E-commerce scenario, iPhone is the fine-grained item while smartphone is the coarse-grained concept to which iPhone belongs. The proposed TDM method leverages this hierarchy of information and turns recommendation problem into a series of hierarchical classification problems. By solving the problem from easy to difficult, TDM can improve both accuracy and efficiency. The main contributions of our paper are summarized as follows:

- To our best knowledge, TDM is the first method that makes arbitrary advanced models possible in generating recommendations from large corpus. Benefiting from hierarchical tree search, TDM achieves logarithmic amount of calculation w.r.t. corpus size when making prediction.
- TDM can help find novel but effective recommendation results more precisely, because the entire corpus is explored and more effective deep models also can help find potential interests.
- Besides more advanced models, TDM also promotes recommendation accuracy by hierarchical search, which divides a large problem into smaller ones and solves them successively from easy to difficult.
- As a kind of index, the tree structure can also be learnt towards optimal hierarchy of items and concepts for more effective retrieval, which in turn facilitates the model training. We employ a tree learning method that allows joint training of neural network and the tree structure.
- We conduct extensive experiments on two large-scale real-world datasets, which show that TDM outperforms existing methods significantly.

It's worth mentioning that tree-based approach is also researched in language model work hierarchical softmax [24], but it's different from the proposed TDM not only in motivation but also in formulation. In next-word prediction problem, conventional softmax has to calculate the normalization term to get any single word's probability, which is very time-consuming. Hierarchical softmax uses tree structure, and next-word's probability is converted to the product of node probabilities along the tree path. Such formulation reduces the computation complexity of next-word's probability to logarithmic magnitude w.r.t. the corpus size. However, in recommendation problem, the goal is to search the entire corpus for those most preferred items, which is a retrieval problem. In hierarchical softmax tree, the optimum of parent nodes can not guarantee that the optimal low level nodes are in their descendants, and all items still need to be traversed to find the optimal one. Thus, it's not suitable for such a retrieval problem. To address the retrieval problem, we propose a max-heap like tree formulation and introduce deep neural networks to model the tree, which forms an efficient method for large-scale recommendation. The following sections will show its difference in formulation and its superiority in performance. In addition, hierarchical softmax adopts a single hidden layer network for a specific natural language processing problem, while the proposed TDM method is practicable to engage any neural network structures.

The proposed tree-based model is a universal solution for all kinds of online content providers. The remainder of this paper is organized as follows: In Section 2, we'll introduce the system architecture of Taobao display advertising to show the position of the proposed method. Section 3 will give a detailed introduction and formalization of the proposed tree-based deep model. And the following Section 4 will describe how the tree-based model serves online. Experimental results on large-scale benchmark dataset and Taobao advertising dataset are shown in Section 5. At last, Section 6 gives our work a conclusion.

2 SYSTEM ARCHITECTURE

In this section, we introduce the architecture of Taobao display advertising recommender system as Figure 1. After receiving page view request from a user, the system uses user features, context features and item features as input to generate a relatively much smaller set (usually hundreds) of candidate items from the entire corpus (hundreds of millions) in the matching server. The tree-based recommendation model takes effort in this stage and shrinks the size of candidate set by several orders of magnitude.

With hundreds of candidate items, the real-time prediction server uses more expressive but also more time consuming models [11, 34] to predict indicators like click-through rate or conversion rate. And after ranking by strategy [17, 35], several items are ultimately impressed to user.

As aforementioned, the proposed recommendation model aims to construct a candidate set with hundreds of items. This stage is essential and also difficult. Whether the user is interested in the generated candidates gives an upper bound of the impression quality. How to draw candidates from the entire corpus weighing efficiency and effectiveness is a problem.

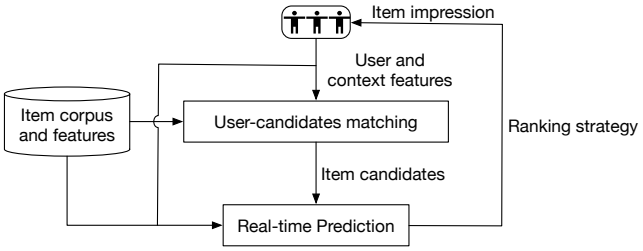


Figure 1: The system architecture of Taobao display advertising recommender system.

3 TREE-BASED DEEP MODEL

In this part, we first introduce the tree structure used in our tree-based model to give an overall conception. Secondly, we introduce hierarchical softmax [24] to show why its formulation is not suitable for recommendation. After that, we give a novel max-heap like tree formulation and show how to train the tree-based model. Then, the deep neural network architecture is introduced. At last, we show how to construct and learn the tree used in the tree-based model.

3.1 Tree for Recommendation

A recommendation tree consists of a set of nodes N, where $N = \{n_1, n_2, \cdots n_{|N|}\}$ represents $|N|$ individual non-leaf or leaf nodes. Each node in N except the root node has one parent and an arbitrary number of children. Specifically, each item c_i in the corpus C corresponds to one and only one *leaf node* in the tree, and those non-leaf nodes are coarse-grained concepts. Without loss of generality, we suppose that node n_1 is always the root node. An example tree is illustrated in the right bottom corner of Figure 2, in which each circle represents a node and the number of node is its index in tree. The tree has 8 leaf nodes in total, each of which corresponds to an item in the corpus. It's worth mentioning that though the given example is a complete binary tree, we don't impose complete and binary as restrictions on the type of the tree in our model.

3.2 Related Work

With the tree structure, we firstly introduce the related work hierarchical softmax to help understand its difference with our TDM. In hierarchical softmax, each leaf node n in tree has its unique encoding from the root to the node. For example, if we encode 1 as choosing the left branch and 0 as choosing the right branch, n_9's encoding in tree in Figure 2 is 110 and n_{15}'s encoding is 000. Denote $b_j(n)$ as the encoding of node n in level j. In hierarchical softmax's formulation, the next-word's probability given the context is derived as

$$P(n|context) = \prod_{j=1}^{w} P\big(b = b_j(n) \big| l_j(n), context\big), \tag{1}$$

where w is the length of leaf node n's encoding, and $l_j(n)$ is n's ancestor node in level j.

In such a way, hierarchical softmax solves the probability calculation problem by avoiding the normalization term (each word in the corpus needs to be traversed) in conventional softmax. However, to find the most possible leaf, the model still has to traverse the entire corpus. Traversing each level's most possible node top-down along the tree path can not guarantee to successfully retrieve the optimal leaf. Therefore, hierarchical softmax's formulation is not suitable for large-scale retrieval problem. In addition, according to Equation 1, each non-leaf node in tree is trained as a binary classifier to discriminate between its two children nodes. But if two nodes are neighbors in the tree, they are probably to be similar. In recommendation scenario, it's likely that user is interested in both two children. Hierarchical softmax's model focuses on distinguishing optimal and suboptimal choices, which may lose the capability of discriminating from a global view. If greedy beam search is used to retrieve those most possible leaf nodes, once bad decisions are made in upper levels of the tree, the model may fail to find relatively better results among those low quality candidates in lower levels. YouTube's work [7] also reports that they have tried hierarchical softmax to learn user and item embeddings, while it performs worse than sampled-softmax [16] manner.

Given that hierarchical softmax's formulation is not suitable for large-scale recommendation, we propose a new tree model formulation in the following section.

3.3 Tree-based Model Formulation

To address the problem of efficient top-k retrieval of most preferred items, we propose a max-heap like tree probability formulation. Max-heap like tree is a tree structure where every non-leaf node n in level j satisfies the following equation for each user u:

$$P^{(j)}(n|u) = \frac{\max\limits_{n_c \in \{n\text{'s children nodes in level } j+1\}} P^{(j+1)}(n_c|u)}{\alpha^{(j)}}, \tag{2}$$

where $P^{(j)}(n|u)$ is the ground truth probability that user u is interested in n. $\alpha^{(j)}$ is the layer-specific normalization term of level j to ensure that the probability sum in the level equals to 1. Equation 2 says that a parent node's ground truth preference equals to the maximum preference of its children nodes, divided by the normalization term. Note that we slightly abuse the notation and let u denote a specific user state. In other words, a specific user state u may transfer to another state u' once the user has a new behavior.

The goal is to find k leaf nodes with largest preference probabilities. Suppose that we have each node n's ground truth $P^{(j)}(n|u)$ in the tree, we can retrieve k nodes with largest preference probabilities layer-wise, and only those children nodes of each level's top k need to be explored. In this way, top k leaf nodes can be ultimately retrieved. Actually, we don't need to know each tree node's exact ground truth probability in the above retrieval process. What we need is the order of the probabilities in each level to help find the top k nodes in the level. Based on this observation, we use user's implicit feedback data and neural network to train each level's discriminator that can tell the order of preference probabilities.

Suppose that user u has an interaction with leaf node n_d, i.e., n_d is a positive sample node for u. It means an order $P^{(m)}(n_d|u) > P^{(m)}(n_t|u)$, where m is the level of leaves and n_t is any other leaf node. In any level j, denote $l_j(n_d)$ as n_d's ancestor in level j. According to the formulation of tree in Equation 2, we can derive that $P^{(j)}(l_j(n_d)|u) > P^{(j)}(n_q|u)$, where n_q is any node in level j

except $l_j(n_d)$. In basis of the above analysis, we can use negative sampling [23] to train each level's order discriminator. In detail, leaf node that have interaction with u, and its ancestor nodes constitute the set of positive samples in each level for u. And randomly selected nodes except positive ones in each level constitute the set of negative samples. Those green and red nodes in Figure 2 give examples for sampling. Suppose that given a user and its state, the target node is n_{13}. Then, n_{13}'s ancestors are positive samples, and those randomly sampled red nodes in each level are negative samples. These samples are then fed into binary probability models to get levels' order discriminators. We use one global deep neural network binary model with different input for all levels' order discriminators. Arbitrary advanced neural network can be adopted to improve model capability.

Denote $\mathcal{Y}_u^+$ and $\mathcal{Y}_u^-$ as the set of positive and negative samples for u. The likelihood function is then derived as:

$$\prod_u \Big(\prod_{n \in \mathcal{Y}_u^+} P\big(\hat{y}_u(n) = 1 | n, u\big) \prod_{n \in \mathcal{Y}_u^-} P\big(\hat{y}_u(n) = 0 | n, u\big) \Big), \quad (3)$$

where $\hat{y}_u(n)$ is the predicted label of node n given u. $P\big(\hat{y}_u(n)|n,u\big)$ is the output of binary probability model, taking user state u and the sampled node n as input. The corresponding loss function is

$$-\sum_u \sum_{n \in \mathcal{Y}_u^+ \cup \mathcal{Y}_u^-} y_u(n) \log P\big(\hat{y}_u(n)=1|n,u\big) + \big(1-y_u(n)\big) \log P\big(\hat{y}_u(n)=0|n,u\big), \quad (4)$$

where $y_u(n)$ is the ground truth label of node n given u. Details about how to train the model according to the loss function are in Section 3.4.

Note that the proposed sampling method is quite different from the underlying one in hierarchical softmax. Compared to the method used in hierarchical softmax which leads the model to distinguish optimal and suboptimal results, we randomly select negative samples in the same level for each positive node. Such method makes each level's discriminator be an intra-level global one. Each level's global discriminator can make precise decisions independently, without depending on the goodness of upper levels' decisions. The global discriminating capability is very important for hierarchical recommendation approaches. It ensures that even if the model makes bad decision and low quality nodes leak into the candidate set in an upper-level, those relatively better nodes rather than very bad ones can be chosen by the model in the following levels.

Given a recommendation tree and an optimized model, the detailed hierarchical prediction algorithm is described in Algorithm 1. The retrieval process is layer-wise and top-down. Suppose that the desired candidate item number is k. For corpus C with size $|C|$, traversing at most $2 * k * \log|C|$ nodes can get the final recommendation set in a complete binary tree. The number of nodes need to be traversed is in a logarithmic relation w.r.t. corpus size, which makes advanced binary probability models possible to be employed.

Our proposed TDM method not only reduces the amount of calculation when making prediction, it also has potential to improve recommendation quality compared with brute-force search in all leaf nodes. Without the tree, training a model to find optimal items directly is a difficult problem because of the corpus size. Employing the tree hierarchy, a large-scale recommendation problem is divided into many smaller problems. There only exist a few nodes in high levels of the tree, thus the discrimination problem is easier. And decisions made by high levels refine the candidate set, which may help lower levels make better judgments. Experimental results in Section 5.4 will show that the proposed hierarchical retrieval approach performs better than direct brute-force search.

Algorithm 1: Layer-wise Retrieval Algorithm in Prediction

Input: User state u, the recommendation tree, the desired item number k, the learnt model
Output: The set of recommended leaf nodes

```
Result set A = ∅, candidate set Q = {the root node n_1};
repeat
    If there are leaf nodes in Q, remove them from Q and
      insert them into A;
    Calculate P(ŷ_u(n) = 1|n, u) for each remaining node
      n ∈ Q;
    Sort nodes in Q in descending order of P(ŷ_u(n) = 1|n, u)
      and derive the set of top k nodes as I;
    Q = {children nodes of n|n ∈ I};
until |Q| == 0;
Return the top k items in set A, according to
  P(ŷ_u(n) = 1|n, u);
```

3.4 The Deep Model

In the following part, we introduce the deep model we use. The entire model is illustrated in Figure 2. Inspired by the click-through rate prediction work [34], we learn low dimensional embeddings for each node in the tree, and use attention module to softly searching for related behaviors for better user representation. To exploit user behavior that contains timestamp information, we design the block-wise input layer to distinguish behaviors that lie in different time windows. The historical behaviors can be divided into different time windows along the timeline, and item embeddings in each time window is weighted averaged. Attention module and the following network greatly strengthen the model capability, and also make user's preferences over candidate items can not be regulated to inner product form.

The embeddings of tree nodes and the tree structure itself are also parts of the model. To minimize Loss 4, the sampled nodes and the corresponding features are used to train the network. Note that we only illustrate the usage of user behavior feature in Figure 2 for briefness, while other features like user profile or contextual feature can be used with no obstacles in practice.

3.5 Tree Construction and Learning

The recommendation tree is a fundamental part of the tree-based deep recommendation model. Unlike multiclass and multi-label classification works [26, 32] where tree is used to partition samples or labels, our recommendation tree indexes items for retrieval. In hierarchical softmax [24], the word hierarchy is built according to

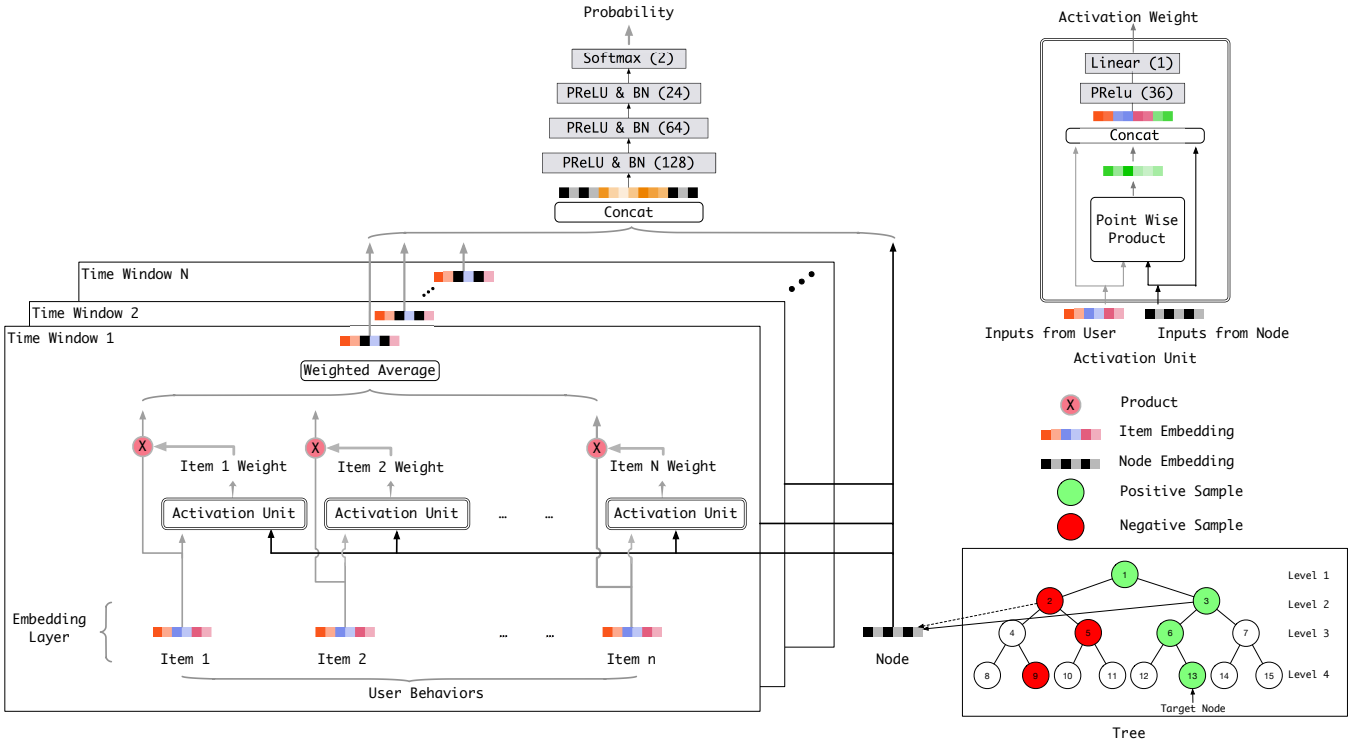


Figure 2: The tree-based deep model architecture. User behaviors are divided into different time windows according to the timestamp. In each time window, item embeddings are weighted averaged, and the weights come from activation units. Each time window's output along with the candidate node's embedding are concatenated as the neural network input. After three fully connected layers with PReLU [33] activation and batch normalization [14], a binary softmax is used to yield the probability whether the user is interested in the candidate node. Each item and its corresponding leaf node share the same embedding. All embeddings are randomly initialized.

expert knowledge from WordNet [21]. In the scenario of recommendation, not every corpus can provide specific expert knowledge. An intuitive alternation is to construct the tree using hierarchical clustering methods in basis of item concurrence or similarity drawn from the dataset. But the clustered tree may be quite imbalanced, which is detrimental for training and retrieval. Given pairwise item similarity, algorithm in [2] gives a way to split items into subsets recursively by spectral clustering [25]. However, spectral clustering is not scalable enough (cubic time complexity w.r.t. corpus size) for large-scale corpus. In this section, we focus on reasonable and feasible tree construction and learning approaches.

Tree initialization. Since we suppose the tree to represent user interests' hierarchical information, it's natural to build the tree in a way that similar items are organized in close positions. Given that category information is extensive available in many domains, we intuitively come up with a method leveraging item's category information to build the initial tree. Without loss of generality, we take binary tree as an example in this section. Firstly, we sort all categories randomly, and place items belonging to the same category together in an intra-category random order. If an item belongs to more than one category, the item is assigned to a random one for uniqueness. In such way, we can get a list of ranked items. Secondly, those ranked items are halved to two equal parts recursively until the current set contains only one item, which could construct a near-complete binary tree top-down. The above kind of category-based initialization can get better hierarchy and results in our experiments than a complete random tree.

Tree learning. As a part of the model, each leaf node's embedding can be learnt after model training. Then we use the learnt leaf nodes' embedding vectors to cluster a new tree. Considering the corpus size, we use k-means clustering algorithm for its good scalability. At each step, items are clustered into two subsets according to their embedding vectors. Note that the two subsets are adjusted to equal for a more balanced tree. The recursion stops when only one item is left, and a binary tree could be constructed in such a top-down way. In our experiments, it takes about an hour to construct such a cluster tree when the corpus size is about 4 millions, using a single machine. Experimental results in Section 5 will show the effectiveness of the given tree learning algorithm.

The deep model and tree structure are learnt jointly in an alternative way: 1) Construct an initial tree and train the model till converging; 2) Learn to get a new tree structure in basis of trained leaf nodes' embeddings; 3) Train the model again with the learnt new tree structure.

4 ONLINE SERVING

Figure 3 illustrates the online serving system of the proposed method. Input feature assembling and item retrieval are split into two asynchronous stages. Each user behavior including click, purchase and adding item into shopping cart will strike the real-time feature server to assemble new input features. And once receiving page view request, the user targeting server will use the pre-assembled features to retrieve candidates from the tree. As described in Algorithm 1, the retrieval is layer-wise and the trained neural network is used to calculate the probability that whether a node is preferred given the input features.

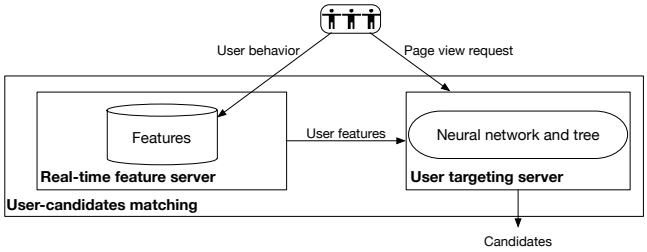


Figure 3: The online serving system of the tree-based model, where user feature is assembled asynchronously.

5 EXPERIMENTAL STUDY

We study the performance of the proposed tree-based model in this section. Experimental results in MovieLens-20M [12] and Taobao advertising dataset called UserBehavior are presented. In the experiments, we compare the proposed method to other existing methods to show the effectiveness of the model, and empirical study results show how the tree-based model and tree learning algorithm work.

5.1 Datasets

The experiments are conducted in two large-scale real-world datasets with timestamps: 1) users' movie viewing data from MovieLens [12]; 2) a user-item behavior dataset from Taobao called UserBehavior. In more details:

MovieLens-20M: It contains user-movie ratings with timestamps in this dataset. As we deal with implicit feedback problem, the ratings are binarized by keeping the ratings of four or higher, which is a common way in other works [8, 20]. Besides, only the users who have watched at least 10 movies are kept. To create training, validation and testing sets, we randomly sample 1, 000 users as testing set and another 1, 000 users as validation set, while the rest users constitute the training set [8]. For validation and testing sets, the first half of user-movie views along the timeline is regarded as known behaviors to predict the latter half.

UserBehavior[1]**:** This dataset is a subset of Taobao user behavior data. We randomly select about 1 million users who have behaviors including click, purchase, adding item to shopping cart and item favoring during November 25 to December 03, 2017. The data is organized in a very similar form to MovieLens-20M, i.e., a user-item behavior consists of user ID, item ID, item's category ID, behavior type and timestamp. As we do in MovieLens-20M, only the users who have at least 10 behaviors are kept. 10, 000 users are randomly selected as testing set and another randomly selected 10, 000 users are validation set. Items' categories are from the bottom level of Taobao's current commodity taxonomy. Table 1 summarizes the major dimensions of the above two datasets after preprocessing.

	MovieLens-20M	UserBehavior
# of users	129,797	969,529
# of items	20,709	4,158,142
# of categories	20	9,436
# of records	9,939,873	100,020,395

Table 1: Dimensions of the two datasets after preprocessing. One record is a user-item pair that represents user feedback.

5.2 Metrics and Comparison Methods

To evaluate the effectiveness of different methods, we use Precision@M, Recall@M and F-Measure@M metrics [20]. Derive the recalled set of items for a user u as $\mathcal{P}_u$ ($|\mathcal{P}_u| = M$) and the user's ground truth set as $\mathcal{G}_u$. Precision@M and Recall@M are

$$Precision@M(u) = \frac{|\mathcal{P}_u \cap \mathcal{G}_u|}{M}, \quad Recall@M(u) = \frac{|\mathcal{P}_u \cap \mathcal{G}_u|}{|\mathcal{G}_u|}, \tag{5}$$

and F-Measure@M is

$$F\text{-}Measure@M(u) = \frac{2 * Precision@M(u) * Recall@M(u)}{Precision@M(u) + Recall@M(u)}. \tag{6}$$

As we emphasize, recommendation results' novelty is responsible for user experience. Existing work [4] gives several approaches to measure the novelty of recommended list of items. Following one of its definition, the Novelty@M is defined as

$$Novelty@M(u) = \frac{|\mathcal{P}_u \setminus \mathcal{S}_u|}{M}, \tag{7}$$

where $\mathcal{S}_u$ is the set of items that have interactions with user u before recommending. User average of the above four metrics in testing set are used to compare the following methods:

- **FM**[28]. FM is a framework for factorization tasks. We use the implementation of FM provided by xLearn[2] project.
- **BPR-MF**[29]. We use its matrix factorization form for implicit feedback recommendation. Implementation of BPR-MF provided by [10] is used.
- **Item-CF**[31]. Item-based collaborative filtering is one of the most widely used personalized recommendation method in production with large-scale corpus [22]. It's also one of the major candidate generation approaches in Taobao. We use the implementation of item-CF provided by Alibaba machine learning platform.
- **YouTube product-DNN**[7] is the deep recommendation approach proposed by YouTube. Sampled-softmax [16] is employed in training, and the inner product of user and item's embeddings reflects the preference. We implement

[1] https://tianchi.aliyun.com/datalab/dataSet.html?spm=5176.100073.0.0.614435eeJVooEG&dataId=649

[2] https://github.com/aksnzhy/xlearn

YouTube product-DNN in Alibaba deep learning platform with the same input features with our proposed model. Exact kNN search in inner product space is adopted in prediction.

- **TDM attention-DNN** (tree-based deep model using attention network) is our proposed method in Figure 2. The tree is initialized in the way described in Section 3.5 and keeps unchanged during the experiments. The implementation is available in GitHub[3].

For FM, BPR-MF and item-CF, we tune several most important hyper-parameters based on the validation set, i.e., the number of factors and iterations in FM and BPR-MF, the number of neighbors in item-CF. FM and BPR-MF require that the users in testing or validation set also have feedback in training set. Therefore, we add the first half of user-item interactions along the timeline in testing and validation set into the training set in both datasets. For YouTube product-DNN and TDM attention-DNN, the node embeddings' dimension is set to 24, because a higher dimension doesn't perform significantly better in our experiments. The hidden unit numbers of three fully connected layers are 128, 64 and 24 respectively. According to the timestamp, user behaviors are divided into 10 time windows. In YouTube product-DNN and TDM attention-DNN, for each implicit feedback we randomly select 100 negative samples in MovieLens-20M and 600 negative samples in UserBehavior. Note that the negative sample number of TDM is the sum of all levels. And we sample more negatives for levels near to leaf.

5.3 Comparison Results

The comparison results of different methods are shown in Table 2 above the dash line. Each metric is the average across all the users in testing set, and the presented values are the average across five different runs for methods with variance.

First, the results indicate that the proposed TDM attention-DNN outperforms all the baselines significantly in both datasets on most of the metrics. Comparing to the second best YouTube product-DNN approach, TDM attention-DNN achieves 21.1% and 42.6% improvements on recall metric in two datasets respectively without filtering. This result proves the effectiveness of advanced neural network and hierarchical tree search adopted by TDM attention-DNN. Among the methods that model user preference over items in inner product form, YouTube product-DNN outperforms BPR-MF and FM because of the usage of neural network. The widely used item-CF method gets worst novelty results, since it has strong memories about what the user has already interacted.

To improve the novelty, a common way in practice is to filter those interacted items in recommendation set [8, 20], i.e., only those novel items could be ultimately recommended. Thus, it's more important to compare accuracy in a complete novel result set. In this experiment, the result set size will be complemented to required number M if its size is smaller than M after filtering. The bottom half of Table 2 shows that TDM attention-DNN outperforms all baselines in large margin as well after filtering interacted items.

To further evaluate the exploration ability of different methods, we do experiments by excluding those interacted categories from recommendation results. Results of each method are also complemented to satisfy the size requirement. Indeed, category-level novelty is currently the most important novelty metric in Taobao recommender system, as we want to reduce the amount of recommendations similar to user's interacted items. Since MovieLens-20M has only 20 categories in total, these experiments are only conducted in UserBehavior dataset and results are shown in Table 3. Take the recall metric for example. We can observe that item-CF's recall is only 1.06%, because its recommendation results can hardly jump out of user's historical behaviors. YouTube product-DNN gets much better results compared to item-CF, since it can explore user's potential interests from the entire corpus. The proposed TDM attention-DNN performs 34.3% better in recall than YouTube's inner product manner. Such huge improvement is very meaningful for recommender systems, and it proves that more advanced model is an enormous difference for recommendation problem.

5.4 Empirical Analysis

Variants of TDM. To comprehend the proposed TDM method itself, we derive and evaluate several variants of TDM:

- **TDM product-DNN.** To find out whether advanced neural network can benefit the results in TDM, we test the variant TDM product-DNN. TDM product-DNN uses the same inner product manner as YouTube product-DNN. Specifically, the attention module in Figure 2 is removed, and the node embedding term is also removed from the network input. The inner product of node embedding and the third fully connected layer's output (without PReLU and BN) along with a sigmoid activation constitute the new binary classifier.
- **TDM DNN.** To further verify the improvements brought by attention module in TDM attention-DNN, we test the variant TDM DNN that only removes the activation unit, i.e., all items' weights are 1.0 in Figure 2.
- **TDM attention-DNN-HS.** As mentioned in Section 3, hierarchical softmax (HS) method [24] is not suitable for recommendation. We test the TDM attention-DNN-HS variant, i.e., use positive nodes' neighbors as negative samples instead of randomly selected ones. Correspondingly, in retrieval of Algorithm 1, the ranking indicator changes from a single node's $P\big(\hat{y}_u(n) = 1|n, u\big)$ to $\prod_{n' \in \text{ n's ancestors}} P\big(\hat{y}_u(n') = 1|n', u\big)$. Attention-DNN is used as the network structure.

The experimental results of the above variants in both datasets are shown in Table 2 under the dash line. Comparing TDM attention-DNN to TDM DNN, the near 10% recall improvement in UserBehavior dataset indicates that the attention module takes impressive efforts. TDM product-DNN performs worse than TDM DNN and TDM attention-DNN, since the inner product manner is much less powerful than the neural network interaction form. These results prove that introducing advanced models in TDM can significantly improve the recommendation performance. Note that TDM attention-DNN-HS gets much worse results compared to TDM attention-DNN, since hierarchical softmax's formulation doesn't fit for recommendation problem.

[3] https://github.com/alibaba/x-deeplearning/tree/master/xdl-algorithm-solution/TDM

Filtering	Method	MovieLens-20M (@10)				UserBehavior (@200)			
		Precision	Recall	F-Measure	Novelty	Precision	Recall	F-Measure	Novelty
None	FM	8.35%	5.12%	5.03%	70.76%	0.31%	1.67%	0.45%	**99.58%**
	BPR-MF	8.10%	5.09%	5.02%	62.56%	0.44%	1.84%	0.64%	99.56%
	Item-CF	8.25%	5.66%	5.29%	59.46%	1.47%	6.95%	2.18%	97.07%
	YouTube product-DNN	11.87%	8.71%	7.96%	71.38%	1.48%	7.58%	2.23%	98.48%
	TDM attention-DNN	**14.06%**	**10.55%**	**9.49%**	**74.15%**	**2.00%**	**10.81%**	**3.03%**	97.30%
	TDM product-DNN	12.20%	9.18%	8.23%	72.78%	1.50%	7.80%	2.26%	98.36%
	TDM DNN	13.35%	10.10%	8.98%	72.18%	1.78%	9.67%	2.70%	97.94%
	TDM attention-DNN-HS	10.92%	9.16%	7.94%	81.00%	1.47%	8.20%	2.25%	98.28%
Interacted items	FM	13.39%	6.87%	7.10%	**100.00%**	0.11%	0.56%	0.17%	**100.00%**
	BPR-MF	13.39%	6.95%	7.17%	**100.00%**	0.36%	1.51%	0.53%	**100.00%**
	Item-CF	15.61%	8.86%	8.81%	**100.00%**	0.68%	4.38%	1.06%	**100.00%**
	YouTube product-DNN	16.51%	10.70%	10.04%	**100.00%**	0.93%	5.67%	1.44%	**100.00%**
	TDM attention-DNN	**17.77%**	**12.31%**	**11.33%**	**100.00%**	**1.16%**	**7.50%**	**1.81%**	**100.00%**
	TDM product-DNN	17.29%	11.87%	10.91%	100.00%	0.92%	5.68%	1.44%	100.00%
	TDM DNN	17.82%	12.12%	11.31%	100.00%	1.02%	6.97%	1.68%	100.00%
	TDM attention-DNN-HS	14.06%	10.72%	9.58%	100.00%	0.86%	5.79%	1.36%	100.00%

Table 2: The comparison results of different methods in MovieLens-20M and UserBehavior datasets. According to the different corpus size, metrics are evaluated @10 in MovieLens-20 and @200 in UserBehavior. In experiments of filtering interacted items, the recommendation results and ground truth only contain items that the user has not yet interacted with before.

Method (@200)	Precision	Recall	F-Measure
Item-CF	0.07%	1.06%	0.13%
YouTube product-DNN	0.26%	3.09%	0.45%
TDM attention-DNN	**0.35%**	**4.15%**	**0.60%**

Table 3: Results in UserBehavior dataset. Items belong to interacted categories are excluded from recommendation results and ground truth.

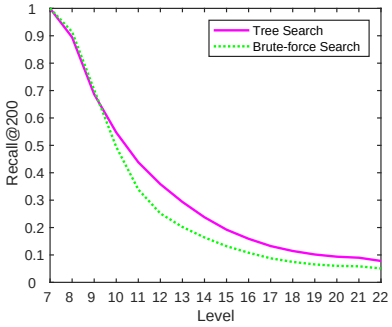


Figure 4: The results of layer-wise Recall@200 in UserBehavior dataset. The ground truth in testing set is traced back to each node's ancestors, till the root node.

Role of the tree. Tree is the key component of the proposed TDM method. It not only acts as an index used in retrieval, but also models the corpus in coarse-to-fine hierarchy. Section 3.3 mentioned that directly making fine-grained recommendation is more difficult than a hierarchical way. We conduct experiments to prove the point of view. Figure 4 illustrates the layer-wise Recall@200 of hierarchical tree search (Algorithm 1) and brute-force search (traverse all nodes in the corresponding level). The experiments are conducted in UserBehavior dataset with TDM product-DNN model, because it's the only variant that is possible to employ brute-force search. Brute-force search slightly outperforms tree search in high levels (level 8, 9), since the node numbers there are small. Once the node number in a level grows, tree search gets better recall results compared to brute-force search, because the tree search can exclude those low quality results in high levels, which reduces the difficulty of the problems in low levels. This result indicates that the hierarchy information contained in the tree structure can help improve recommendation preciseness.

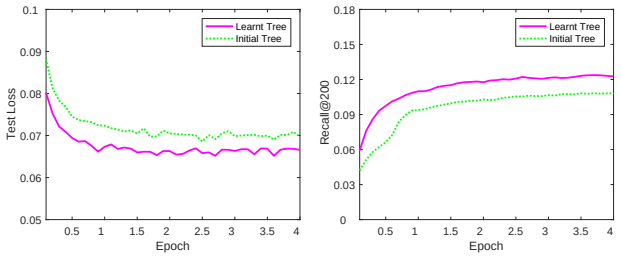
Tree learning. In Section 3.5, we propose the tree initialization and learning algorithms. Table 4 gives the comparison results between initial tree and learnt tree. From the results, we can observe that the trained model with learnt tree structure significantly outperforms the initial one. For example, the recall metric of learnt tree increases from 4.15% to 4.82% compared to initial tree in experiments of filtering interacted categories, which surpasses YouTube product-DNN's 3.09% and item-CF's 1.06% in very large margin. To further compare these two trees, we illustrate the test loss and recall curve of TDM attention-DNN method w.r.t. training iterations in Figure 5. From Figure 5(a), we can see that the learnt tree structure gets smaller test loss. And both Figure 5(a) and 5(b) indicate that the model converges to better results with learnt tree. The above results prove that the tree learning algorithm can improve the hierarchy of items, further to facilitate training and prediction.

5.5 Online Results

We evaluate the proposed TDM method in Taobao display advertising platform with real traffic. The experiments are conducted in *Guess What You Like* column of Taobao App Homepage. Two online metrics are used to measure the performance: click-through

Filtering	Tree	Precision	Recall	F-Measure	Novelty
None	Initial	2.00%	10.81%	3.03%	**97.30%**
	Learnt	**2.34%**	**12.37%**	**3.54%**	96.68%
Interacted items	Initial	1.16%	7.50%	1.81%	**100.00%**
	Learnt	**1.33%**	**8.38%**	**2.09%**	**100.00%**
Interacted categories	Initial	0.35%	4.15%	0.60%	**100.00%**
	Learnt	**0.40%**	**4.82%**	**0.69%**	**100.00%**

Table 4: Comparison results of different tree structures in UserBehavior dataset using TDM attention-DNN model (@200). Tree is initialized and learnt according to the algorithm described in Section 3.5.



(a) Test Loss (b) Test Recall

Figure 5: The test loss and test Recall@200 on UserBehavior dataset for initial and learnt tree.

rate (CTR) and revenue per mille (RPM). Details are as follows:

$$CTR = \frac{\text{\# of clicks}}{\text{\# of impressions}}, RPM = \frac{\text{Ad revenue}}{\text{\# of impressions}} * 1000. \quad (8)$$

In our advertising system, advertisers bid on some given ad clusters. There are about 1.4 million clusters and each ad cluster contains hundreds or thousands of similar ads. The experiments are conducted in the granularity of ad cluster to keep consistent with the existing system. The comparison method is mixture of logistic regression [9] that used to pick out superior results only from those interacted clusters, which is a strong baseline. Since there are many stages in the system like CTR prediction [11, 34] and ranking [35] as illustrated in Figure 1, deploying and evaluating the proposed TDM method online is a huge project, which involves the linkage and optimization of the whole system. We have finished the deployment of the first TDM DNN version so far and evaluated its improvements online. Each of the comparison buckets has 5% of all online traffic. It's worth mentioning that there are several online simultaneously running recommendation methods. They take efforts in different point of views, and their recommendation results are merged together for the following stages. TDM only replaces the most effective one of them while keeping other modules unchanged. The average metric lift rates of the testing bucket with TDM are listed in Table 5.

As shown in Table 5, the CTR of TDM method increases 2.1%. This improvement indicates that the proposed method can recall more accurate results for users. And on the other hand the RPM metric increases 6.4%, which means the TDM method can also bring more revenue for Taobao advertising platform. TDM has been deployed to serve major online traffic, we believe that the above improvement is only a preliminary result in a huge project, and there has room for further improvements.

Metric	CTR	RPM
Lift Rate	2.1%	6.4%

Table 5: Online results from Jan 22 to Jan 28, 2018 in *Guess What You Like* column of Taobao App Homepage.

Prediction efficiency. TDM makes advanced neural network feasible to interact user and items in large-scale recommendation, which opens a new perspective of view in recommender systems. It's worth mentioning that though advanced neural networks need more calculation when inferring, but the complexity of a whole prediction process is no larger than $O(k * \log|C| * t)$, where k is the required results size, $|C|$ is the corpus size and t is the complexity of network's single feed-forward pass. This complexity upper bound is acceptable under current CPU/GPU hardware conditions, and user side's features are shared across different nodes in one retrieval and some calculation could be shared according to model designs. In Taobao display advertising system, it actually takes the deployed TDM DNN model about 6 milliseconds to recommend once in average. Such running time is shorter than the following click-through rate prediction module, and is not the system's bottleneck.

6 CONCLUSION

We figure out the main challenge for model-based methods to generate recommendations from large-scale corpus, i.e., the amount of calculation problem when making prediction. A tree-based approach is proposed, where arbitrary advanced models can be employed in large-scale recommendation to infer user interests coarse-to-fine along the tree. Besides training the model, a tree structure learning approach is used, which proves that a better tree structure can lead to significantly better results. A possible future direction is to design more elaborate tree learning approaches. We conduct extensive experiments which validate the effectiveness of the proposed method, both in recommendation accuracy and novelty. In addition, empirical analysis showcases how and why the proposed method works. In Taobao display advertising platform, the proposed TDM method has been deployed in production, which improves both business benefits and user experience.

ACKNOWLEDGEMENTS

We deeply appreciate Jian Xu, Chengru Song, Chuan Yu, Guorui Zhou and Yongliang Wang for their helpful suggestions and discussions. Thank Huimin Yi, Yang Zheng, Zelin Hu, Sui Huang, Yin Yang and Bochao Liu for implementing the key components of the training and serving infrastructure. Thank Haiyang He, Yangyang Fu and Yang Wang for necessary engineering supports.

REFERENCES

[1] Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. 2013. Multi-label learning with millions of labels: Recommending advertiser bid phrases for

web pages. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 13–24.

[2] Samy Bengio, Jason Weston, and David Grangier. 2010. Label embedding trees for large multi-class tasks. In *International Conference on Neural Information Processing Systems*. 163–171.

[3] Alina Beygelzimer, John Langford, and Pradeep Ravikumar. 2007. Multiclass classification with filter trees. *Gynecologic Oncology* 105, 2 (2007), 312–320.

[4] Pablo Castells, SaÃžl Vargas, and Jun Wang. 2011. Novelty and Diversity Metrics for Recommender Systems: Choice, Discovery and Relevance. *In Proceedings of International Workshop on Diversity in Document Retrieval* (2011), 29–37.

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 7–10.

[6] Anna E Choromanska and John Langford. 2015. Logarithmic time online multiclass prediction. In *Advances in Neural Information Processing Systems*. 55–63.

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *ACM Conference on Recommender Systems*. 191–198.

[8] Robin Devooght and Hugues Bersini. 2016. Collaborative filtering with recurrent neural networks. *arXiv preprint arXiv:1608.07400* (2016).

[9] Kun Gai, Xiaoqiang Zhu, Han Li, Kai Liu, and Zhe Wang. 2017. Learning Piecewise Linear Models from Large Scale Data for Ad Click Prediction. *arXiv preprint arXiv:1704.05194* (2017).

[10] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2011. MyMediaLite: A free recommender system library. In *Proceedings of the fifth ACM conference on Recommender systems*. ACM, 305–308.

[11] Tiezheng Ge, Liqin Zhao, Guorui Zhou, Keyu Chen, Shuying Liu, Huiming Yi, Zelin Hu, Bochao Liu, Peng Sun, Haoyu Liu, et al. 2017. Image Matters: Jointly Train Advertising CTR Model with Image Representation of Ad and User Behavior. *arXiv preprint arXiv:1711.06505* (2017).

[12] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems* 5, 4 (2016), 19.

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

[14] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. 448–456.

[15] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. 2016. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 935–944.

[16] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2014. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007* (2014).

[17] Junqi Jin, Chengru Song, Han Li, Kun Gai, Jun Wang, and Weinan Zhang. 2018. Real-Time Bidding with Multi-Agent Reinforcement Learning in Display Advertising. *arXiv preprint arXiv:1802.09756* (2018).

[18] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734* (2017).

[19] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37.

[20] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M. Blei. 2016. Factorization Meets the Item Embedding:Regularizing Matrix Factorization with Item Co-occurrence. In *ACM Conference on Recommender Systems*. 59–66.

[21] D. Lin. 1999. WordNet: An Electronic Lexical Database. *Computational Linguistics* 25, 2 (1999), 292–296.

[22] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.

[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *International Conference on Neural Information Processing Systems*. 3111–3119.

[24] Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. *Aistats* (2005).

[25] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. 2001. On spectral clustering: analysis and an algorithm. In *International Conference on Neural Information Processing Systems: Natural and Synthetic*. 849–856.

[26] Yashoteja Prabhu and Manik Varma. 2014. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 263–272.

[27] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *IEEE 16th International Conference on Data Mining*. IEEE, 1149–1154.

[28] Steffen Rendle. 2010. Factorization Machines. In *IEEE International Conference on Data Mining*. 995–1000.

[29] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th conference on uncertainty in artificial intelligence*. AUAI Press, 452–461.

[30] Ruslan Salakhutdinov and Andriy Mnih. 2007. Probabilistic Matrix Factorization. In *International Conference on Neural Information Processing Systems*. 1257–1264.

[31] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *International Conference on World Wide Web*. 285–295.

[32] J. Weston, A. Makadia, and H. Yee. 2013. Label partitioning for sublinear ranking. In *International Conference on Machine Learning*. 181–189.

[33] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv:1505.00853* (2015).

[34] Guorui Zhou, Chengru Song, Xiaoqiang Zhu, Xiao Ma, Yanghui Yan, Xingya Dai, Han Zhu, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD Conference*. ACM.

[35] Han Zhu, Junqi Jin, Chang Tan, Fei Pan, Yifan Zeng, Han Li, and Kun Gai. 2017. Optimized Cost Per Click in Taobao Display Advertising. In *Proceedings of the 23rd ACM SIGKDD Conference*. ACM, 2191–2200.