arXiv:2007.07203v1 [cs.IR] 12 Jul 2020

Deep Retrieval: An End-to-End Learnable Structure Model for Large-Scale Recommendations

Weihao Gao[*], **Xiangjun Fan**[*], **Jiankai Sun**, **Kai Jia**
Bytedance Inc.
{weihao.gao, xiangjun.fan, jiankai.sun, jiakai}@bytedance.com

Wenzhi Xiao, **Chong Wang**, **Xiaobing Liu**
Bytedance Inc.
{xiaowenzhi, chong.wang, will.liu}@bytedance.com

Abstract

One of the core problems in large-scale recommendations is to retrieve top relevant candidates accurately and efficiently, preferably in sub-linear time. Previous approaches are mostly based on a two-step procedure: first learn an inner-product model and then use maximum inner product search (MIPS) algorithms to search top candidates, leading to potential loss of retrieval accuracy. In this paper, we present Deep Retrieval (DR), an end-to-end learnable structure model for large-scale recommendations. DR encodes all candidates into a discrete latent space. Those latent codes for the candidates are model parameters and to be learnt together with other neural network parameters to maximize the same objective function. With the model learnt, a beam search over the latent codes is performed to retrieve the top candidates. Empirically, we showed that DR, with sub-linear computational complexity, can achieve almost the same accuracy as the brute-force baseline.

1 Introduction

Recommendation systems have gained great success in various commercial applications for decades. The objective of recommendation systems is to retrieve relevant candidates from an enormous corpus based on user features and historical behaviors. In the era of mobile internet, the amount of candidates from content providers and the amount of active users both grow rapidly to tens of millions to hundred of millions, making it more challenging to design accurate recommendation systems. The scalability and efficiency of the algorithm are the main challenge for modern recommendation systems.

One of the early successful techniques of recommendation systems is the collaborative filtering (CF), which makes predictions based on the simple idea that similar users may prefer similar items. Item-based collaborative filtering (Item-CF) [21] extends the idea by considering the similarities between items and items, and later lays a foundation for Amazon's recommendation system [14].

Recently, vector-based recommendation algorithms have been widely adopted. The main idea is to embed users and items in a latent vector space, and use the inner product of vectors to represent the preference between users and items. Representative vector embedding methods includes matrix factorization (MF) [17, 12], factorization machines (FM) [20], DeepFM [5], Field-aware FM (FFM) [11], etc. However, when the number of items is large, the complexity of brute-force computation of the inner product for all items can be prohibitive. Thus, maximum inner product search (MIPS) or approximate nearest neighbors (ANN) algorithms are usually used to retrieve items

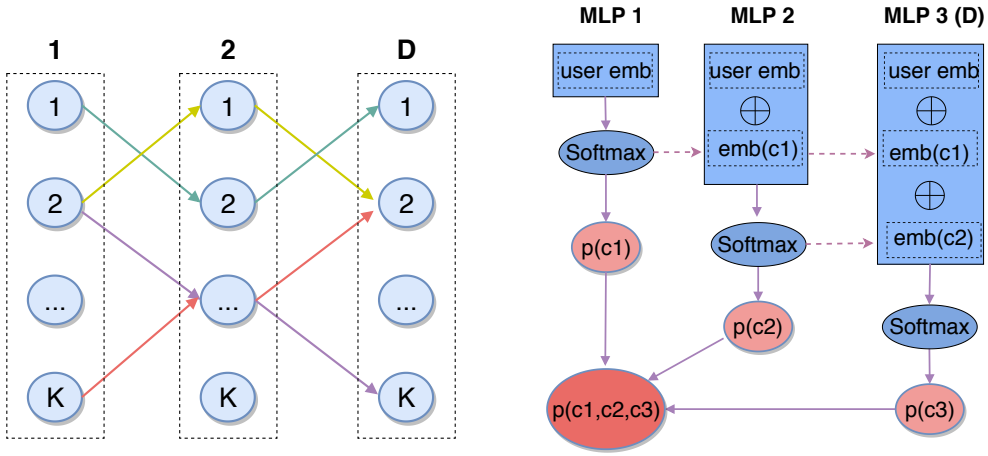
[*]Co-first authors.

Preprint. Under review.

when the corpus is large. Efficient MIPS or ANN algorithms includes tree-based algorithms [18, 9], locality sensitive hashing (LSH) [22, 23], product quantization (PQ) [10, 4], hierarchical navigable small world graphs (HNSW) [15], etc.

Despite their success in real world applications, vector-based algorithms has two main deficiencies: (1) The objective of learning vector representation and learning good MIPS structure are not perfectly aligned; (2) The dependency on inner products of user and item embeddings limits the capability of the model [8]. In order to break these limitations, tree based models [24, 25] have been proposed. These methods use a tree as indices and map each item to a leaf node of the tree. Learning objectives for model parameters and tree structures are well aligned to improve the accuracy. However, the tree structure itself can be difficult to learn: data available at the leaf level can be scarce and might not provide enough signal to learn a good tree at that finer level.

In this paper, we proposed an end-to-end trainable structure model — Deep Retrieval (DR). Instead of using a tree structure, we propose to use a $D \times K$ matrix as in Figure 1a for indexing, motivated by [2]. Each item is indexed by one or more "codes" (or equivalently "paths") with length D and range $\{1, 2, \ldots, K\}$. For example, an item related to chocolate may be encoded by $[36, 204, 105]$ and another item related to cake may be encoded by $[36, 227, 120]$. There are K^D possible paths and each path can be interpreted as a cluster of items: each path could contain multiple items and each item could also belong to multiple paths.



(a) (b)

Figure 1: (a) Consider a structure with width $K = 100$ and depth $D = 3$. Assuming an item is encoded by length-D vector $[36, 27, 20]$, which is called a "code" or a "path". The path denotes that the item is assigned to the $(1, 36)$, $(2, 27)$, $(3, 20)$ indices of the $K \times D$ matrix. In the figure, arrows with the same color form a path. Different paths could intersect with each other by sharing the same index at some layer. (b) Flow chart showing the process for constructing the probability $p(c|x, \theta)$.

The advantage of DR is two-folded. In training, the item paths can be learnt together with the neural network parameters of the structure model using an expectation-maximization (EM) type algorithm. Therefore the entire training process is end-to-end and can be easily deployed on deep learning platforms. In terms of model capability, the multiple-to-multiple encoding scheme enables DR to learn more complicated relationships among users and items. Experiments show the benefit of being able to learn such complicated relationships. The rest of the paper is organized as follows.

- In Section 2, we describe the structure model and the structure objective function used in training in detail. We also introduce a beam search algorithm to find candidate paths in the inference stage.
- In Section 3, we introduce the EM algorithm for training neural network parameters and paths of items jointly. We adopt a penalization on the sizes of the paths to prevent overfitting.

- In Section 4, we demonstrate the performance of DR on two public datasets: MovieLens-20M[2] and Amazon books[3]. Experiment results show that DR can almost achieve the brute-force accuracy with sub-linear computational complexity.
- In Section 5, we conclude the paper and discuss several possible future research directions.

2 Structure Model

In this section, we introduce the structure model in Deep Retrieval in detail. Firstly, we construct the probability for user x_i to select path c_i given the model parameters θ and present the training objective. Then we introduce a multi-path mechanism to enable the model to capture multi-aspect properties of items. In the inference stage, we introduce a beam search algorithm to select candidate paths based on user embeddings.

2.1 Structure Objective Function

The structure model contains D layers [4] with K nodes in each layer, where each layer is a multi-layer perceptron (MLP) with skip connections and softmax as output. Each layer takes an input vector and outputs a probability distribution over $\{1, 2, \ldots, K\}$ based on parameters θ. Let $[V] = \{1, \ldots, V\}$ be the labels of all items and $\pi : V \to [K]^D$ be the mapping from items to paths. We assume π is fixed in this section and will introduce the algorithm for learning π together with θ in Section 3.

Given a pair of training sample (x, y), which denotes a positive interaction (click, convert, like, etc.) between user x and item y, as well as the path $c = (c_1, c_2, \ldots, c_D)$ associated with the item y, the probability $p(c|x, \theta)$ is constructed layer by layer as follows (see Figure 1b for a flow chart).

- The first layer takes the user embedding $\mathrm{emb}(x)$ as input, and output a probability $p(c_1|x, \theta_1)$ over the K nodes of the first layer, based on parameters θ_1 .
- From the second layer onward, we concatenate the user embedding $\mathrm{emb}(x)$ and the embeddings of all the previous layers $\mathrm{emb}(c_{d-1})$ (called path embeddings) as the input of MLP, and output $p(c_d|x, c_1, \ldots, c_{d-1}, \theta_d)$ over the K nodes of layer d, based on parameters θ_d.
- The probability of $p(c|x, \theta)$ is just the product of the probabilities of all the layers.

$$p(c|x, \theta) = \prod_{d=1}^{D} p(c_d|x, c_1, \ldots, c_{d-1}, \theta_d). \tag{1}$$

Given a set of N training samples $\{(x_i, y_i)\}_{i=1}^N$, we maximize the log likelihood function of the structure model.

$$\begin{aligned} \mathcal{Q}_{\mathrm{str}}(\theta, \pi) &= \sum_{i=1}^{N} \log p(c_i = \pi(y_i)|x_i, \theta) \\ &= \sum_{i=1}^{N} \sum_{d=1}^{D} \log p(c_{i,d}|x_i, c_{i,1}, \ldots, c_{i,d-1}, \theta_d). \end{aligned} \tag{2}$$

The size of the input vector of layer d is the embedding size times d, and the size of the output vector is K. The parameters of layer d have a size of $\Theta(Kd)$. The parameters θ contain the parameters θ_d's in all layers, as well as the path embeddings. The parameters in the entire model have an order of $\Theta(KD^2)$, which is significantly smaller than the number of possible paths K^D when $D \geq 2$.

2.2 Multi-path Structure Objective

In tree-based deep models [25, 24] as well as the structure model we introduced before, each item belongs to only one cluster, which limits the capacity of the model to express multi-aspect information in real data. For example, an item related to kebab should belong to a cluster related to food. An item

[2] `https://grouplens.org/datasets/movielens`

[3] `http://jmcauley.ucsd.edu/data/amazon`

[4] Other neural network architectures such as recurrent neural networks (RNN) can also be applied here. Since D is not very large in our settings, for simplicity, we use MLP here.

related to flowers should belong to a cluster related to gifts. However, an item related to chocolate or cakes should belong to both clusters in order to be recommended to users interested in either food or gifts. In real world recommendation systems, a cluster might not have an explicit meaning such as food or gifts, but this example motivates us to assign each item to multiple clusters. In DR, we allow each item y_i to be assigned to J different paths $\{c_{i,1}, \ldots, c_{i,J}\}$. Let $\pi : V \to [K]^{D \times J}$ be the mapping from items to multiple paths and the multi-path structure objective is defined as,

$$\mathcal{Q}_{\text{str}}(\theta, \pi) = \sum_{i=1}^{N} \log \left(\sum_{j=1}^{J} p(c_{i,j} = \pi_j(y_i) | x_i, \theta) \right). \tag{3}$$

Beam search for inference. In the inference stage, we want to retrieve items from the structure model, given user embeddings as input. To this end, we utilize beam search algorithm [19] to retrieve multiple paths and merge the items in the retrieved paths. In each layer, the algorithm selects top B nodes from the all successors of the selected nodes from the previous layer. Finally it returns B top paths in the final layer. The time complexity of the inference stage is $O(DKB \log B)$, which is sub-linear with respect to the number of items. The detail of the beam search algorithm is relegated to the appendix.

3 Learning

In the previous section, we introduced the structure model in Deep Retrieval and the structure objective to be optimized. The objective is entirely continuous with respect to the parameters, hence can be optimized by any gradient-based optimizer. However, the objective involves the mapping from items to paths π, which is discrete and can not be optimized by gradient-based optimizer. This mapping acts as the "clustering" of items, which motivates us to use an EM-style algorithm to optimize the mapping and the continuous parameters jointly. In this section, we describe the EM algorithm in detail and introduce a penalization term to prevent overfitting.

3.1 EM Algorithm for Joint Training

Given a user-item pair (x_i, y_i) in training data set, let the path associate with the item $(\{\pi_1(y_i), \ldots, \pi_J(y_i)\})$ be the latent data in EM algorithm. Along with the continuous parameters θ, the objective function is given by

$$\begin{aligned} \mathcal{Q}_{\text{str}}(\theta, \pi) &= \sum_{i=1}^{N} \log \left(\sum_{j=1}^{J} p(\pi_j(y_i) | x_i, \theta) \right) \\ &= \sum_{v=1}^{V} \left(\sum_{i: y_i = v} \log \left(\sum_{j=1}^{J} p(\pi_j(v) | x_i, \theta) \right) \right). \end{aligned} \tag{4}$$

We maximize the objective function over all possible mappings π. However, there are K^D number of possible paths so we could not maximize it over all $\pi_j(v)$'s. Instead, we only record the values for the top paths using beam search and leave the rest paths with zero scores[5]. Unfortunately, this renders an unstable function due to the presence of $\log(0)$ in Equation (4). To address this, we approximate it using an upper bound [6] $\sum_{i=1}^{N} \log p_i \le N(\log \sum_{i=1}^{N} p_i - \log N)$ to obtain

$$\mathcal{Q}_{\text{str}}(\theta, \pi) \le \overline{\mathcal{Q}}_{\text{str}}(\theta, \pi) = \sum_{v=1}^{V} \left(N_v \log \left(\sum_{j=1}^{J} \sum_{i: y_i = v} p(\pi_j(v) | x_i, \theta) \right) - \log N_v \right), \tag{5}$$

where $N_v = \sum_{i=1}^{N} \mathbb{I}[i : y_i = v]$ denotes the number of occurrences of v in the training set which is independent of the mapping. We denote the score $s[v, c]$ as $\sum_{i: y_i = v} p(c | x_i, \theta)$ for simplicity

[5] We also tried to use a small value. But it is difficult to figure out what the small value should be.

[6] Since this is an upper bound for the true objective we are maximizing, there is no guarantee as to maximizing a surrogate via a lower bound. However we still find it works well in practice.

of notation. In practice, it is impossible to retain all scores as the possible number of paths c is exponentially large, so we only retain a subset of S paths with larger scores through beam search. In M-step, we simply maximize $\sum_{j=1}^{J} s[v, \pi_j(v)]$ over each $\pi_j(v)$, which is equivalent to pick J highest scores of $s[v, c]$ among the top paths c from beam search. The EM training algorithm is summarized in Algorithm 1.

Algorithm 1 EM algorithm for structure learning

Input: training set $\{x_i, y_i\}_{i=1}^{N}$. T is some predefined epoch number.
Initialize $\pi_j^{(0)}$ and θ randomly.
for $t = 1$ **to** T **do**
  Fixed $\pi^{(t-1)}$, optimize parameter θ using a gradient-based optimizer to maximize structure objective $\mathcal{Q}_{\text{str}}(\theta, \pi^{(t-1)})$.
  for $v = 1$ **to** V **do**
    Compute $s[v, c]$ among the top paths c from beam search.
    Let $\{\pi_1^{(t)}(v), \ldots, \pi_J^{(t)}(v)\} \leftarrow$ top J scores of $s[v, c]$.
  end for
end for

3.2 Penalization on Size of Paths

Overfitting is likely to happen if we do not apply any penalization for Algorithm 1. Imagine that the structure model is overfitted and gives a particular path $(0, 0, 0)$ a very high probability, for any input. Then in M-step, all the items will be assigned to path $(0, 0, 0)$, making the model fail to cluster the items. In order to prevent overfitting, we introduce a penalization term on the size of the paths. The penalized $\mathcal{Q}$ function is given by

$$\begin{aligned}\mathcal{Q}_{\text{pen}}(\theta, \pi) &= \overline{\mathcal{Q}}_{\text{str}}(\theta, \pi) - \alpha \cdot \sum_{c \in [K]^D} f(|c|) \\ &= \sum_{v=1}^{V} \left(N_v \log \left(\sum_{j=1}^{J} s[v, \pi_j(v)] \right) - \log N_v \right) - \alpha \cdot \sum_{c \in [K]^D} f(|c|). \end{aligned} \tag{6}$$

where α is the penalty factor, $|c|$ denotes the number of items in c and f is an increasing and convex function. A quadratic function $f(|c|) = |c|^2/2$ controls the average size of paths, and higher order polynomials penalize more on larger paths. In our experiments, we use $f(|c|) = |c|^4/4$. It's worth mentioning that the penalty is only applied in M-step, not during training of continuous parameters θ.

Coordinate descent algorithm for path assignment. It is intractable to jointly optimize the penalized objective function (6) over all the path assignments, since the penalization term can not be decomposed into summation of terms of each item. So we use coordinate descent algorithm in M-step. We fix the assignment of all other items while optimizing the path assignments $\pi_j(v)$'s of v. The time complexity for coordinate descent is $O(VJS)$, where S is the pool size for candidate paths. The detail of the coordinate descent algorithm is relegated to the appendix.

3.3 Multi-task Learning and Reranking with Softmax Models

Based on the experiments we conducted in this paper, we found that jointly training DR with a softmax classification model greatly improves the performance. We conjecture that this is because the paths for the items are randomly assigned in the beginning, leading to increased difficulty for optimization. By sharing the inputs with an easy-to-train softmax model, we are able to give the structure model an uplift in the optimization direction. So the final objective we are maximizing is

$$\mathcal{Q} = \mathcal{Q}_{\text{str}} + \mathcal{Q}_{\text{softmax}}.$$

After performing beam search to retrieval a set of candidate items using Algorithm 3, we use the softmax function to rerank those candidates to obtain the final top candidates.

3.4 Complexity Analysis

We summarize the time complexity of each stage in Deep Retrieval as follows. The time complexity of the inference stage is $O(DKB \log B)$ per sample, which is sub-linear with respect to the number of items. The time complexity of training continuous parameters θ is $O(NJKD^2)$, where N is the number of training samples in one epoch and J be the multiplicity of paths. The time complexity of path assignment by coordinate descent algorithm is $O(VJS)$, where V is corpus size and S is the pool size of candidate paths.

4 Experiments

In this section, we study the performance of DR on two public recommendation datasets: MovieLens-20M [6] and Amazon books [7, 16]. We compare the performance of DR with brute-force algorithm, as well as several other recommendation baselines including tree-based models TDM [25] and JTM [24]. In the end of this section, we investigate the role of important hyperparameters in DR.

4.1 Datasets and Metrics

MovieLens-20M. This dataset contains rating and free-text tagging activities from a movie recommendation service called MovieLens. We use the 20M subset which were created by the behaviors of 138,493 users between 1995 and 2015. Each user-movie interaction contains a used-id, a movie-id, a rating between 1.0 to 5.0, as well as a timestamp.

In order to make a fair comparison, we exactly follow the same data pre-processing procedure as TDM. We only keep records with rating higher or equal to 4.0, and only keep users with at least ten reviews. After pre-processing, the dataset contains 129,797 users, 20,709 movies and 9,939,873 interactions. Then we randomly sample 1,000 users and corresponding records to construct the validation set, another 1,000 users to construct the test set, and other users to construct the training set. For each user, the first half of the reviews according to the timestamp are used as historical behavior features and the latter half are used as ground truths to be predicted.

Amazon books. This dataset contains user reviews of books from Amazon, where each user-book interaction contains a user-id, an item-id, and corresponding timestamp. Similar to MovieLens-20M, we follow the same pre-processing procedure as JTM. The dataset contains 294,739 users, 1,477,922 items and 8,654,619 interactions. Notice that Amazon books dataset have much more items but sparser interactions than MovieLens-20M. We randomly sample 5,000 users and corresponding records as the test set, another 5,000 users as the validation set and other users as the training set. The construction procedures of behavior features and ground truths are the same as in MovieLens-20M.

Metrics. We use precision, recall and F-measure as metrics to evaluate the performance for each algorithm. We emphasize that the metrics are computed for each user individually, and averaged without weight across users, following the same setting as both TDM and JTM. We compute the metrics by retrieving top 10 and 200 items for each user in MovieLens-20M and Amazon books respectively.

Model and training. Here we present some details about the model and training procedure in the experiment. Since the dataset is split in a way such that the users in the training set, validation set and test set are disjoint, we drop the user-id and only use the behavior sequence as input for DR. The behavior sequence is truncated to length of 69 if it is longer than 69, and filled with a placeholder symbol if it is shorter than 69. A recurrent neural network with GRU is utilized to project the behavior sequence onto a fixed dimension embedding as the input of DR. We adopt the multi-task learning framework, and rerank the items in the recalled paths by a softmax reranker. We train the embeddings of DR and softmax jointly for the initial two epochs, freeze the embeddings of softmax and train the embeddings of DR for two more epochs. The reason is to prevent overfitting of the softmax model. In the inference stage, the number of items retrieved from beam search is not fixed due to the differences of path sizes, but the variance is not large. Empirically we control the beam size such that the number of items from beam search is 5 to 10 times the number of finally retrieved items.

Algorithm	Precision@10	Recall@10	F-measure@10
Item-CF	8.25%	5.66%	5.29%
YouTube DNN	11.87%	8.71%	7.96%
TDM (best)	14.06%	10.55%	9.49%
DR	$20.58\% \pm 0.47\%$	$10.89\% \pm 0.32\%$	$12.32\% \pm 0.36\%$
Brute-force	$20.70\% \pm 0.16\%$	$10.96\% \pm 0.32\%$	$12.38\% \pm 0.32\%$

Table 1: Comparison of precision@10, recall@10 and F-measure@10 for DR, brute-force retrieval and other recommendation algorithms on MovieLens-20M.

Algorithm	Precision@200	Recall@200	F-measure@200
Item-CF	0.52%	8.18%	0.92%
YouTube DNN	0.53%	8.26%	0.93%
TDM (best)	0.56%	8.57%	0.98%
JTM	0.79%	12.45%	1.38%
DR	$0.95\% \pm 0.01\%$	$13.74\% \pm 0.14\%$	$1.63\% \pm 0.02\%$
Brute-force	$0.95\% \pm 0.01\%$	$13.75\% \pm 0.10\%$	$1.63\% \pm 0.02\%$

Table 2: Comparison of precision@200, recall@200 and F-measure@200 for DR, brute-force and other recommendation algorithms on Amazon Books.

4.2 Empirical Results

We compare the performance of DR with the following algorithms: Item-CF [21], YouTube product DNN [3], TDM and JTM. We directly use the numbers of Item-CF, Youtube DNN, TDM and JTM from TDM and JTM papers for fair comparison. Among the different variants of TDM presented, we pick the one with best performance. The result of JTM is only available for Amazon books. We also compare DR with brute-force retrieval algorithm, which directly computes the inner-product of user embedding and all the item embeddings learnt in the softmax model and returns the top K items. The brute-force algorithm is usually computationally prohibitive in practical large recommendation systems, but can be used as an upper bound for small dataset for inner-product based models.

Table 1 shows the performance of DR compared to other algorithms and brute-force for MovieLens-20M. Table 2 shows the results for Amazon books. For DR and brute-force, we independently train the same model for 5 times and compute the mean and standard deviation of each metric. We conclude the following results: (1) DR performs better than other methods including tree-based retrieval algorithms such as TDM and JTM. (2) The performance of DR is very close to or on par with the performance of brute-force method.

4.3 Sensitivity of Hyperparameters

DR introduces some key hyperparameters which may infect the performance dramatically, including the width of the structure model K, number of multiple paths J, beam size B and penalty factor α. In the MovieLens-20M experiment, we choose $K = 50$, $B = 25$, $J = 3$ and $\alpha = 3 \times 10^{-5}$. In the Amazon books experiment, we choose $K = 100$, $B = 50$, $J = 3$ and $\alpha = 3 \times 10^{-7}$.

Using the Amazon books dataset, we show the role of these hyperparameters and see how they may affect the performance. We present how the recall@200 change as these hyperparameters change in Figure 2. We fix the value of other hyperparameters unchanged when varying one hyperparameter. Precision@200 and F-measure@200 follow similar trends so we plot them in the appendix.

- **Width of model** K controls the overall capacity of the model. If K is too small, the number of clusters is too small for all the items; if K is too big, the time complexity of training and inference stages grow linearly with K. Moreover, large K may increase the possibility of overfitting. An appropriate K should be chosen depending on the size of the corpus.
- **Number of paths** J enables the model to express multi-aspect information of items. The performance is the worst when $J = 1$, and keeps increasing as J increases. Large J may

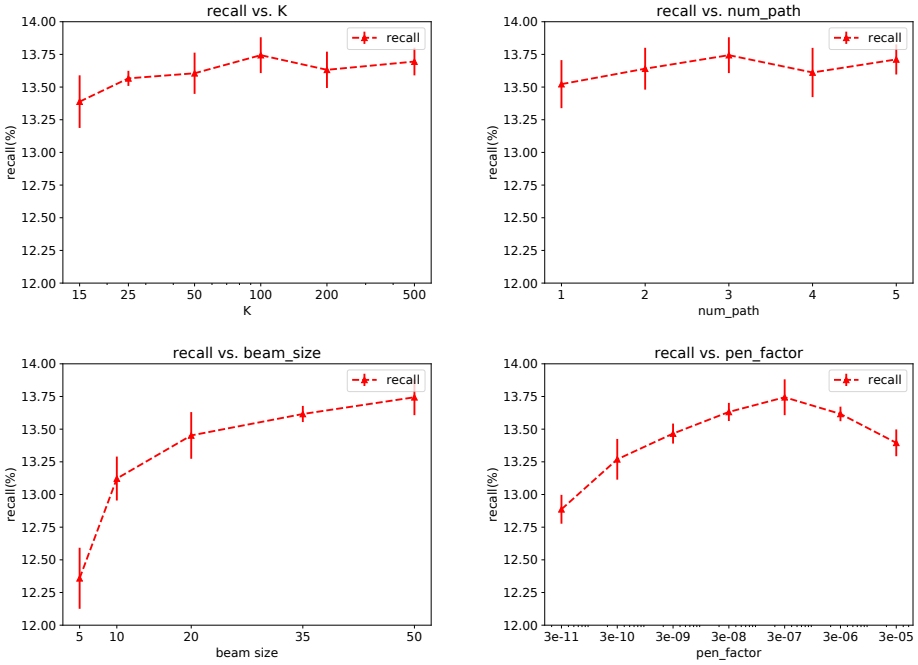


Figure 2: Relationship between recall@200 in Amazon Books experiment and model width K, number of paths J, beam size B and penalty factor α, respectively.

Penalty factor	3e-10	3e-9	3e-8	3e-7	3e-6
Top path size	3837 ± 197	1948 ± 30	956 ± 29	459 ± 13	242 ± 1

Table 3: Relationship between the path with most items (called top path) and penalty factor α

not affect the performance, but the time complexity for training grows linearly with J. In practice, choosing J between 3 and 5 is recommended.

- **Beam size** B controls the number of candidate paths to be recalled. Larger B leads a better performance as well as heavier computation in the inference stage.
- **Penalty factor** α controls the number of items in each path. The best performance is achieved when α falls in a certain range. Smaller α leads to a larger path size (see Table 3) hence heavier computation in the reranking stage. Beam size B and penalty factor α should be appropriately chosen as a trade off between model performance and inference speed.

Overall, we can see that DR is fairly stable to hyperparameters since there is a wide range of hyperparameters which leads to near-optimal performances.

5 Conclusion and Discussion

In this paper, we have proposed Deep Retrieval, an end-to-end learnable structure model for large-scale recommender systems. DR uses an EM-style algorithm to learn the model parameters and paths of items jointly. Experiments have shown that DR performs well compared with brute-force baselines in two public recommendation datasets.

There are several future research directions based on the current model design. Firstly, the structure model defines the probability distribution on paths only based on user side information. An useful idea would be how to incorporate item side information more directly into the DR model. Secondly, we only make use of positive interactions such as click, convert or like between user and item. Negative interactions such as non-click, dislike or unfollow should also be considered in future work to improve the model performance. Finally, the current DR still uses a softmax model as a reranker,

which might have an upper bound of the performance capped by the softmax model. We are actively working to address this issue as well.

Broader Impact

It is widely believed that practical recommender systems not only reflect user preferences, they often shape them over time [13, 1]. And this can lead to potential biases on decision making or user behaviors due to the continuous feedback loop in the system. Our proposed method is more on the perspective of improving the core machine learning technology to better retrieve top relevant candidates based on historical user behavior data. Whether this method amplifies or mitigates the existing biases in real recommender systems needs to be further examined.

References

[1] Gediminas Adomavicius, Jesse Bockstedt, Shawn P Curley, Jingjing Zhang, and Sam Ransbotham. The hidden side effects of recommendation systems. *MIT Sloan Management Review*, 60(2):1, 2019.

[2] Ting Chen, Martin Renqiang Min, and Yizhou Sun. Learning k-way d-dimensional discrete codes for compact embedding representations. *arXiv preprint arXiv:1806.09464*, 2018.

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

[4] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence*, 36(4):744–755, 2013.

[5] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

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

[7] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*, pages 507–517, 2016.

[8] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[9] Michael E Houle and Michael Nett. Rank-based similarity search: Reducing the dimensional dependence. *IEEE transactions on pattern analysis and machine intelligence*, 37(1):136–150, 2014.

[10] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[11] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 43–50, 2016.

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

[13] Sanjay Krishnan, Jay Patel, Michael J Franklin, and Ken Goldberg. A methodology for learning, analyzing, and mitigating social influence bias in recommender systems. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 137–144, 2014.

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

[15] Yury A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[16] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–52, 2015.

[17] Andriy Mnih and Russ R Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[18] Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2227–2240, 2014.

[19] D Raj Reddy et al. Speech understanding systems: A summary of results of the five-year research effort. *Department of Computer Science. Camegie-Mell University, Pittsburgh, PA*, 17, 1977.

[20] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.

[21] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295, 2001.

[22] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pages 2321–2329, 2014.

[23] Ryan Spring and Anshumali Shrivastava. A new unbiased and efficient class of lsh-based samplers and estimators for partition function computation in log-linear models. *arXiv preprint arXiv:1703.05160*, 2017.

[24] Han Zhu, Daqing Chang, Ziru Xu, Pengye Zhang, Xiang Li, Jie He, Han Li, Jian Xu, and Kun Gai. Joint optimization of tree-based index and deep model for recommender systems. In *Advances in Neural Information Processing Systems*, pages 3973–3982, 2019.

[25] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1079–1088, 2018.

Appendix

A Coordinate descent algorithm for penalized path assignment

In this section, we illustrate the coordinate descent algorithm used in path assignment with penalty in detail. Recall that in penalized M-step, we want to maximize the following objective over all assignments $\{\pi_j(v)\}_{j=1}^J$'s.

$$\arg\max_{\{\pi_j(v)\}_{j=1}^J} \sum_{v=1}^V \left(N_v \log \left(\sum_{j=1}^J s[v, \pi_j(v)] \right) - \log N_v \right) - \alpha \cdot \sum_{c \in [K]^D} f(|c|). \tag{7}$$

Now we apply the coordinate descent algorithm by fixing the assignments of all other items and focus on item v. Notice that the term $-\log N_v$ is irrelevant to $\pi_j(v)$ hence can be dropped. For each item v, the partial objective function can be written as

$$\arg\max_{\pi_1(v),\ldots,\pi_J(v)} N_v \log \left(\sum_{j=1}^J s[v, \pi_j(v)] \right) - \alpha \sum_{j=1}^J f(|\pi_j(v)|), \tag{8}$$

The coordinate descent algorithm is given as follows. In practise, three to five iterations are enough to ensure the algorithm converges. The time complexity grows linear with vocabulary size V, multiplicity of paths J as well as number of candidate paths S.

Algorithm 2 Coordinate descent algorithm for penalized path assignment

Input: Score functions $\log s[v, c]$. Number of iterations T.
Initialize $|c| = 0$ for all paths c.
for $t = 1$ **to** T **do**
  for all items v **do**
    $\text{sum} \leftarrow 0$.
    for $j = 1$ **to** J **do**
      if $t > 1$ **then**
        $|\pi_j^{(t-1)}(v)| \leftarrow |\pi_j^{(t-1)}(v)| - 1$.
      end if
      for all candidate paths c of item v such that $c \notin \{\pi_l^{(t)}(v)\}_{l=1}^{j-1}$ **do**
        Compute penalized scores

$$\tilde{s}[v, c] = \log\left(s[v, c] + \text{sum}\right) - \alpha\left(f(|c| + 1) - f(|c|)\right).$$

      end for
      $\pi_j^{(t)}(v) \leftarrow \arg\max_c \tilde{s}[v, c]$.
      $\text{sum} \leftarrow \text{sum} + s[v, \pi_j^{(t)}(v)]$.
      $|\pi_j^{(t)}(v)| \leftarrow |\pi_j^{(t)}(v)| + 1$.
    end for
  end for
end for
Output: path assignments $\{\pi_j^{(T)}(v)\}_{j=1}^J$.

B Beam search algorithm for inference

In this section, we present the beam search algorithm for inference in detail. Given user embedding x and structure model with parameter θ, the beam search algorithm (1) picks the top B nodes at the first layer; (2) picks the top B nodes among the successors of the chosen nodes at the previous layer; (3) outputs the final B nodes at the final layer. The algorithm is shown in Algorithm 3. In each layer, choosing top B from $K \times B$ candidates has a time complexity of $O(KB \log B)$. The total complexity is $O(DKB \log B)$.

Algorithm 3 Beam search algorithm

Input user x, structure model with parameter θ, beam size B.
Let $C_1 = \{c_{1,1}, \ldots, c_{1,B}\}$ be top B entries of $\{p(c_1|x,\theta) : c_1 \in \{1, \ldots, K\}\}$.
for $d = 2$ to D **do**
Let $C_d = \{(c_{1,1}, \ldots, c_{d,1}), \ldots, (c_{1,B}, \ldots, c_{d,B})\}$ be the top B entries of the set of all successors of C_{d-1} defined as follows.

$$\{p(c_1, \ldots, c_{d-1}|x,\theta)p(c_d|x, c_1, \ldots, c_{d-1}, \theta) : (c_1, \ldots, c_{d-1}) \in C_d, c_d \in \{1, \ldots, K\}\}.$$

end for
Output C_D, a set of B paths.

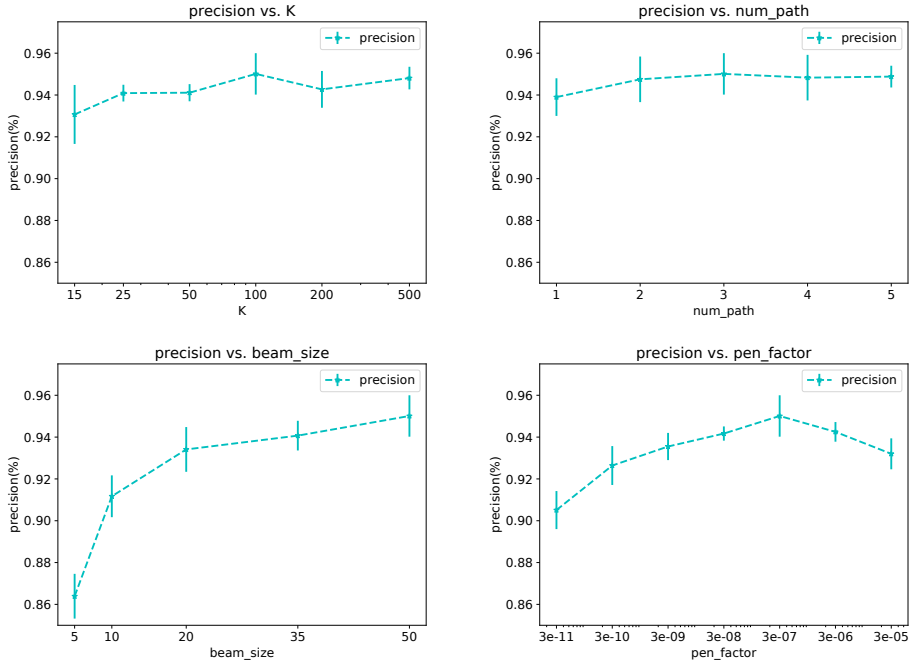


Figure 3: Relationship between precision@200 in Amazon Books experiment and model width K, number of paths J, beam size B and penalty factor α, respectively.

C Precision and F-measure against hyperparameters for Amazon books experiment

Here we plot the precision@200 and F-measure@200 against hyperparameters in the Amazon books datasets. The results of precision@200 are shown in Figure 3 and the results of F-measure@200 are shown in Figure 4. We can see that both the precision and the F-measure follow the same trend as the recall shown in Section 4.3.

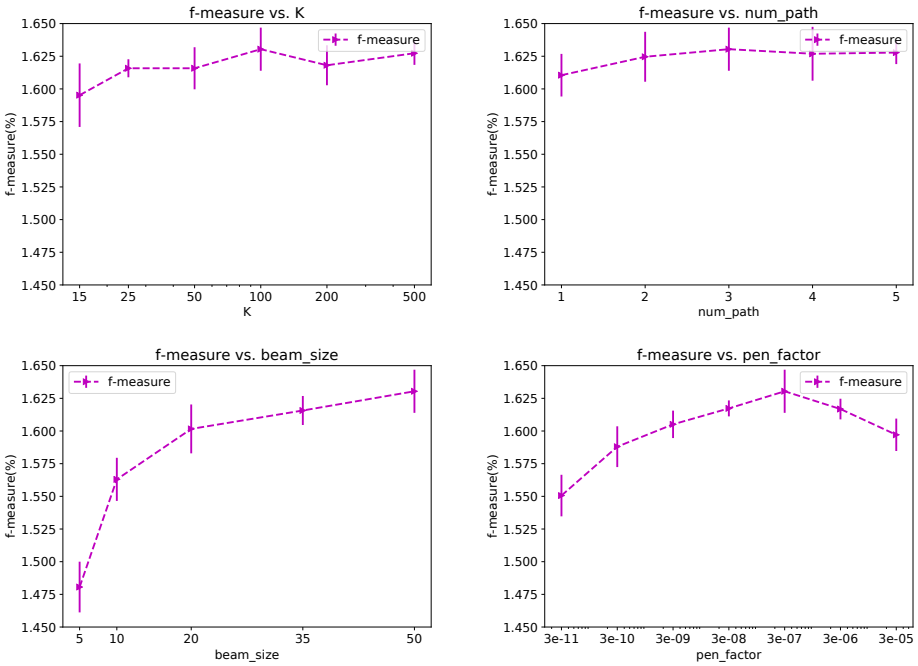


Figure 4: Relationship between F-measure@200 in Amazon Books experiment and model width K, number of paths J, beam size B and penalty factor α, respectively.