Continuous Input Embedding Size Search For Recommender Systems

Yunke Qu
The University of Queensland
Brisbane, Australia
yunke.qu@uq.net.au

Tong Chen
The University of Queensland
Brisbane, Australia
tong.chen@uq.edu.au

Xiangyu Zhao
City University of Hong Kong
Hong Kong
xy.zhao@cityu.edu.hk

Lizhen Cui
Shandong University
Jinan, China
clz@sdu.edu.cn

Kai Zheng
School of Computer Science and
Engineering and Shenzhen Institute
for Advanced Study, University of
Electronic Science and Engineering of
China
Chengdu, China
zhengkai@uestc.edu.cn

Hongzhi Yin*
The University of Queensland
Brisbane, Australia
h.yin1@uq.edu.au

ABSTRACT

Latent factor models are the most popular backbones for today's recommender systems owing to their prominent performance. Latent factor models represent users and items as real-valued embedding vectors for pairwise similarity computation, and all embeddings are traditionally restricted to a uniform size that is relatively large (e.g., 256-dimensional). With the exponentially expanding user base and item catalog in contemporary e commerce, this design is admittedly becoming memory-inefficient. To facilitate lightweight recommendation, reinforcement learning (RL) has recently opened up opportunities for identifying varying embedding sizes for different users/items. However, challenged by search efficiency and learning an optimal RL policy, existing RL-based methods are restricted to highly discrete, predefined embedding size choices. This leads to a largely overlooked potential of introducing finer granularity into embedding sizes to obtain better recommendation effectiveness under a given memory budget. In this paper, we propose continuous input embedding size search (CIESS), a novel RL-based method that operates on a continuous search space with arbitrary embedding sizes to choose from. In CIESS, we further present an innovative random walk-based exploration strategy to allow the RL policy to efficiently explore more candidate embedding sizes and converge to a better decision. CIESS is also model-agnostic and hence generalizable to a variety of latent factor recommender systems, whilst experiments on two real-world datasets have shown state-of-the-art performance of CIESS under different memory budgets when

paired with three popular recommendation models. Code is available at https://github.com/qykcq/Continuous-Input-Embedding-Size-Search-For-Recommender-Systems.

CCS CONCEPTS

• **Information systems** → **Recommender systems**.

KEYWORDS

recommender systems, reinforcement learning

1 INTRODUCTION

Recommender systems predict a user's preference for an item based on their previous interactions with other items [47, 56] and have been widely applied in various e-commerce services. As arguably the most representative and powerful recommendation algorithm, latent factor models use an embedding table to map user and item IDs to dedicated vector representations (i.e., embeddings). The user and item embeddings are then fed into a pairwise similarity function (e.g., dot product or deep neural networks) to predict a user's preference for each item. However, the embedding tables can be memory-inefficient and pose challenges for storage [59] and deployment on personal devices [4, 23, 46] due to the large number of users and items in modern applications.

The root cause is that, each user/item embedding in the conventional embedding table shares the same fixed embedding size. An example from [24] shows that a recommender model embedding 10 million items into 256-dimensional vectors can exceed 9 GB memory consumption in a double-precision float system. As such, researchers have developed better solutions to compress the embedding table while maintaining its expressiveness. The most primitive

*Corresponding author

recommender uses binary codes as an alternative to embeddings [55], which was soon supplanted by another line of methods that compress fixed-sized embeddings into a lightly parameterized component. For example, codebooks have been proposed to store the embedding latent vectors efficiently in [24, 38, 52]. [51] introduced the semi-tensor product operation to tensor-train decomposition to derive an ultra-compact embedding table, and [18] encoded users and items into hash codes and applied neural layers to learn their dense representations. However, these methods must be redesigned and retrained for different memory budgets to maintain optimal recommendation performance.

Due to this inflexibility, there has been a new line of methods featuring reinforcement learning (RL) to automatically search for variable-size embeddings while balancing the memory consumption. For instance, in [26], a policy network was used to select the embedding size for each user/item from a predefined set of actions. [17] first discretizes the embedding table and then devised a RL-based policy network that searches the optimal embedding size configuration. RL allows us to plug in a memory cost term in the reward function to adaptively adjust the embedding sizes to achieve an ideal trade-off between space complexity and accuracy. Despite improved performance, these methods are built on a highly discrete search space with a small collection of predefined embedding sizes (e.g., only six choices in [26]. Consequently, such a narrow range of choices do not necessarily contain the optimal size for each user/item. This is likely to result in suboptimal solutions since the suggested embedding sizes may be either too small to guarantee expressiveness of important users'/items' representations, or too huge to be memory-efficient.

Ideally, an embedding search paradigm should allow each user/item to have arbitrary embedding sizes, maximizing the potential of obtaining optimal performance. However, non-trivial challenges have to be addressed before we can allow for such freedom of candidate embedding sizes in the search space. Firstly, although a straightforward solution is to discretize the action space by treating every integer in the interval (e.g., $[1, 256]$) as a candidate size, the policy network will be prone to suboptimal effectiveness due to the vast action space [25, 54]. Secondly, such conversions will also pose challenges on training efficiency. On the one hand, the commonly used greedy exploration, i.e., iteratively selecting the action with the maximum expected reward will involve the costly "train-and-evaluate" cycle under every possible embedding size, which can quickly become computationally prohibitive in the recommendation setting. On the other hand, despite the possible remedies from learning parameterized functions to estimate an action's quality values (i.e., Q-values) [7], training and evaluating such parameterized functions still requires sufficient coverage of the embedding sizes assigned to different users/items, which again brings back the performance bottleneck.

In light of these challenges, we propose continuous input embedding size search (CIESS), which is an RL-based algorithm that can efficiently operate in a (near[1]) continuous action space for embedding size search. To enable generalization to a continuous action space, we build CIESS upon an actor-critic paradigm [10], specifically the twin delayed deep deterministic policy gradient

(TD3) [8], where we have designed a policy/actor network that can determine the best embedding size from an interval based on the state of each user/item. Compared with existing embedding size search counterparts [17, 26, 58] that only support discrete embedding size search from a small pool of actions, this is the first work to explore a large, continuous, and fine-grained RL search space of embedding sizes, thus unlocking the full potential of learning a compact recommender system with optimal recommendation performance. Furthermore, CIESS is a versatile embedding size search approach that does not hold any assumptions on the backbone recommendation model, and is compatible with a variety of latent factor recommenders that require an embedding table.

However, given the large number of possible embedding sizes, it is unlikely the actor will always reach the states with the highest reward, introducing high variance in the estimation of the optimal Q-value. In short, when applied to continuous embedding size search, the actor in TD3 will be tasked to maximize Q-values computed by the parameterized estimator (i.e., the critic) that is hard to train and potentially erroneous, leading to inferior performance. In CIESS, we propose to explore a group of candidate embedding sizes, and select the one with the maximum Q-value in each iteration. To achieve this without greedily evaluating all possible actions, we innovatively design a *random walk* mechanism in our actor-critic optimization. By performing random walks from the original action produced by the actor networks, the actors sample a small sequence of alternative actions similar to the current one. Next, the actor passes this sequence of actions to the critic for selecting the most rewarding action, which is in turn used to optimize the actor. Intuitively, this brings controlled mutations to currently the best embedding size selected, and pushes CIESS to explore better choices with higher rewards. We will empirically show that the random walk component endows the actor with improved convergence, and hence stronger utility of the resulted embedding table after compression.

To sum up, our work entails the following contributions:

- We point out that relaxing the discrete and narrow action space into a continuous one with arbitrary dimensionality choices yields better expressiveness of compressed embeddings for RL-based recommender embedding size search.
- We propose CIESS, a novel embedding size search method with RL. CIESS innovatively operates on a continuous interval to locate the best embedding size for each user/item, where a random walk-based actor-critic scheme is designed to guarantee optimal embedding size decisions amid the substantially enlarged action space.
- We conduct extensive experimental comparisons with state-of-the-art baselines paired with a variety of base recommenders, where the results have verified the advantageous efficacy of CIESS.

2 METHODOLOGY

CIESS has two main components that work alternately during training: (1) a recommendation model $F_\Theta(\cdot)$ parameterized by Θ; and (2) the RL-based search function $G_\Phi(\cdot)$ parameterized by Φ. A schematic view of CIESS's workflow is shown in Figure 1. In each optimization iteration of CIESS, the recommender $F_\Theta(\cdot)$ adjusts

---

[1]This is due to the fact that embedding sizes can only be integers.

its user/item embedding sizes to the ones provided by the policy $G_\Phi(\cdot)$, then updates its parameters Θ with training samples. Afterwards, $F_\Theta(\cdot)$ is evaluated on a hold-out dataset, where the top-k recommendation quality can be measured by common metrics such as Recall at Rank k (Recall@k) and Normalized Discounted Cumulative Gain at Rank k (NDCG@k). Based on the recommendation quality, the search function $G_\Phi(\cdot)$ will be revised, and then updates its embedding size selection for each user/item for the next iteration. In what follows, we unfold the design of CIESS.

2.1 Base Recommender with Masked Embeddings

Let $\mathcal{U}$ and $\mathcal{V}$ be a set of users u and items v, respectively. Their embedding vectors are stored in a real-valued embedding table $\mathbf{E}$ with the dimensionality of $(|\mathcal{U}| + |\mathcal{V}|) \times d_{max}$. It can be viewed as the vertical concatenation of all user and item embeddings $[\mathbf{e}_{u_1}; ...; \mathbf{e}_{u_{|\mathcal{U}|}}; \mathbf{e}_{v_1}; ...; \mathbf{e}_{v_{|\mathcal{V}|}}]$, where d_{max} is the initial embedding dimension of all users/items in the full-size embedding table. In other words, d_{max} is also the maximum embedding size in the search space.

By performing embedding look-ups, we can map each user/item ID to a real-valued embedding vector $\mathbf{e}_u$ and $\mathbf{e}_v$. To enable adjustable embedding sizes, we introduce a binary mask $\mathbf{M} \in \{0, 1\}^{(|\mathcal{U}|+|\mathcal{V}|) \times d_{max}}$, which is applied to $\mathbf{E}$ during the embedding look-up:

$$\mathbf{e}_n = \text{Lookup}(\mathbf{E} \odot \mathbf{M}, n), \quad n \in \mathcal{U} \cup \mathcal{V}, \quad (1)$$

where $\odot$ is the element-wise multiplication and n is the ID of a user/item. **For simplicity, we use $n \in \mathcal{U} \cup \mathcal{V}$ to index either a user/item when there is no ambiguity**. Notably, $\mathbf{M}$ is dynamically updated according to the current policy to control the usable dimensions of each embedding vector. Given an automatically learned embedding size d_n for a specific user/item, the s-th element of its corresponding mask vector $\mathbf{m}_n$ is defined as:

$$\mathbf{m}_{n(s)} = \begin{cases} 1 & \text{for } 1 \leq s \leq d_n \\ 0 & \text{for } d_n < s < d_{max} \end{cases}, \quad n \in \mathcal{U} \cup \mathcal{V}. \quad (2)$$

With the mask $\mathbf{M}$, for each user/item, we can retain the first d_n elements of its full embedding while setting all succeeding dimensions to 0. It is worth noting that, performing embedding sparsification by masking unwanted dimensions with zeros is a commonly adopted approach in lightweight recommender systems [27, 30], as the resulting embedding table can take advantage of the latest sparse matrix storage techniques [37, 45] that bring negligible cost for storing zero-valued entries.

On obtaining the sparsified embeddings of both users and items, the recommendation model $F_\Theta(\cdot)$ will output a preference score $\hat{y}_{uv}$ denoting the pairwise user-item affinity:

$$\hat{y}_{uv} = F_\Theta(\mathbf{e}_u, \mathbf{e}_v), \quad u \in \mathcal{U}, v \in \mathcal{V}, \quad (3)$$

where the choice of $F_\Theta(\cdot)$ can be arbitrary, as long as it supports such pairwise similarity computation with user and item embeddings.

Objective w.r.t. Embedding Size Search. For optimizing the recommender, we adopt the well-established Bayesian Personalized
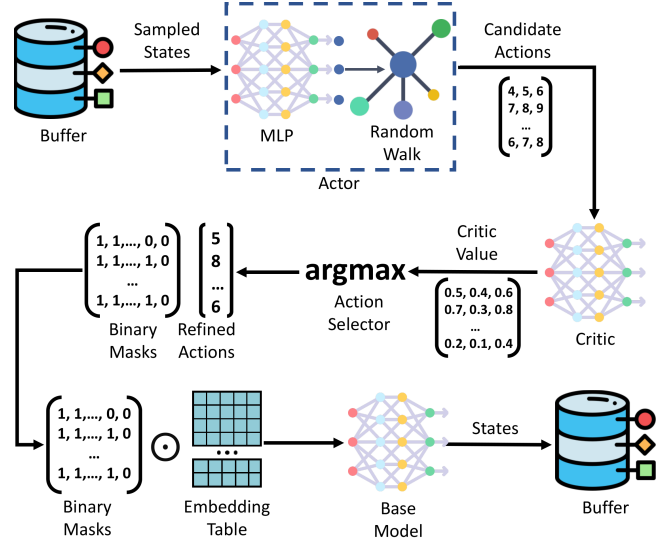

Figure 1: An overarching view of CIESS.

Ranking (BPR) Loss [36]:

$$\mathcal{L}_{BPR} = \sum_{(u,v,v') \in D} -\ln \sigma(\hat{y}_{uv} - \hat{y}_{uv'}) + \gamma\|\Theta\|_2^2, \quad (4)$$

where $\mathcal{D}$ denotes the training dataset, (u, v, v') denotes the user u prefers item v over item v', and $\hat{y}_{uv}$ and $\hat{y}_{uv'}$ are the predicted preferences that the user u has for items v and v'. The second term is the $L2$ regularization weighted by γ for overfitting prevention. As we are interested in performing embedding size search for each individual user and item under a given memory budget, we define the overall objective as follows:

$$\min_{\Theta,\Phi} \mathcal{L}_{BPR} \quad \text{s.t. } 1 - \frac{\|\mathbf{M}\|_{1,1}}{(|\mathcal{U}| + |\mathcal{V}|) \times d_{max}} \geq c, \quad (5)$$

where our target sparsity ratio c ($0 < c < 1$) specifies the percentage of parameters to be pruned (i.e., zeroed out in our case) in the final sparse embedding matrix $\mathbf{E} \odot \mathbf{M}$.

2.2 Continuous Embedding Size Search with Reinforcement Learning

Now that the base recommender can accommodate varying embedding sizes via masked sparsification, we start to search for the optimal embedding sizes with $G_\Phi(\cdot)$. In order to efficiently learn a quality embedding size search policy from a continuous action space, we hereby introduce our solution in an RL setting by presenting our design of the environment, state, action, reward, actor and critic.

2.2.1 Environment. As discussed in Section 2.1, the base recommender model allows for adjustable embedding sizes. During the optimization process, the environment receives the action (i.e., embedding sizes for all user/items), provides feedback (i.e., reward) on both the memory cost and recommendation performance, and update its state for the subsequent action prediction.

2.2.2 State. The state s is the input to the policy network (i.e., actor in CIESS) that drives the decision-making on each user/item-specific embedding size. [58] shows that the popularity (i.e., the number of interactions) and the current embedding size decision of the user/item are effective in providing the policy network the context for subsequent search. Our method inherits this design, with an additional quality indicator a that records the recommendation accuracy fluctuation under the current policy:

$$s_n = (\frac{f_n - f_{min}}{f_{max} - f_{min}}, \frac{d_n - d_{min}}{d_{max} - d_{min}}, q_n), \quad n \in \mathcal{U} \cup \mathcal{V}, \quad (6)$$

where f_n is the popularity of the user/item, normalized by the corresponding maximum/minimum frequency among all users $f_{max} = \max_{u \in \mathcal{U}}(f_u)/f_{min} = \min_{u \in \mathcal{U}}(f_u)$ or items $f_{max} = \max_{v \in \mathcal{V}}(f_v)/f_{min} = \min_{v \in \mathcal{V}}(f_v)$, which are observed from training data. d_n is the current embedding size allocated to a user/item, and q_n quantifies the changes in the recommendation quality when the embedding size decreases from d_{max} to the current d_n. Compared with [58], incorporating this quality indicator into the state is able to help trace the impact from the most recent action (i.e., embedding size) to the recommendation effectiveness, which can encourage the policy network to better balance the memory cost and performance with the embedding sizes selected from a vast, continuous action space.

For q_n, we define it as the ratio between the current ranking quality under the currently selected embedding size d_n and the ranking quality under the initial/maximal embedding size d_{max}:

$$q_n = \min(\frac{\text{eval}(\mathbf{e}_n | \mathbf{E} \odot \mathbf{M})}{\text{eval}(\mathbf{e}_n | \mathbf{E})}, 1), \quad n \in \mathcal{U} \cup \mathcal{V}, \quad (7)$$

where $\text{eval}(\cdot)$ evaluates the recommendation performance w.r.t. a user/item embedding $\mathbf{e}_n$ drawn from the specified embedding table. With the $\min(\cdot)$ operator, we restrict $q_n \in [0, 1]$. The denominator $\text{eval}(\mathbf{e}_n | \mathbf{E})$ can be precomputed with the fully trained based recommender and reused throughout the search process. Instead of using the raw recommendation accuracy measure $\text{eval}(\mathbf{e}_n | \mathbf{E} \odot \mathbf{M})$, we use the ratio format in Eq.(7) to indicate the fluctuation (mostly reduction) of recommendation quality when the embeddings are compressed. This is due to the fact that some users and items are inherently trivial or hard to rank, e.g., long-tail users tend to have a relatively lower recommendation accuracy even with full embeddings, and adjusting their embedding sizes will not significantly affect the accuracy obtained. In such cases, using the performance ratio prevents misleading signals and amplifies the reward for those users/items.

Since Recall@k and NDCG@k are common metrics [13, 20] for recommendation evaluation, we implement an ensemble of both Recall and NDCG scores under different k values for $\text{eval}(\cdot)$:

$$\text{eval}(\mathbf{e}_u | \cdot) = \frac{\sum_{k \in \mathcal{K}} \text{Recall}@k_u + \text{NDCG}@k_u}{2|\mathcal{K}|}, \quad u \in \mathcal{U}, \quad (8)$$

where the choices of k in our paper are $\mathcal{K} = \{5, 10, 20\}$, and Recall@$k_u$ and NDCG@k_u denote the evaluation scores w.r.t. user $u \in \mathcal{U}$.

Note that, Eq.(8) is only applicable to user embeddings, as recommendation metrics are essentially user-oriented. Hence, for item $v \in \mathcal{V}$ and its embedding $\mathbf{e}_v$, we identify its interacted users $\mathcal{U}_v$ in the training set, then take the average of all $\text{eval}(\mathbf{e}_u | \cdot)$ scores for

$u \in \mathcal{U}_v$:

$$\text{eval}(\mathbf{e}_v | \cdot) = \frac{1}{|\mathcal{U}_v|} \sum_{u \in \mathcal{U}_v} \text{eval}(\mathbf{e}_u | \cdot), \quad v \in \mathcal{V}, \quad (9)$$

where $\mathbf{e}_u$ and $\mathbf{e}_v$ are drawn from the same embedding table.

2.2.3 Reward. The reward r is the feedback to the current policy to guide subsequent embedding size adjustments. Given our goal stated in Eq.(5), the reward should reflect a blend of recommendation quality and space complexity. Therefore, we design the following pointwise reward for a user/item on their current embedding size d_u / d_v selected:

$$r_n = q_n - \lambda(\frac{d_n}{d_{max}})^2, \quad n \in \mathcal{U} \cup \mathcal{V}, \quad (10)$$

where the first term is the ranking quality defined in Eq.(7), and the second term weighted by the scaling hyperparameter λ measures the memory cost of the embedding size chosen. The squared form in the memory cost magnifies the reward gain during early pruning stages (e.g., when $\frac{d_n}{d_{max}}$ drops from 1 to 0.9) and stabilizes that at later stages (e.g., when $\frac{d_n}{d_{max}}$ drops from 0.2 to 0.1). As such, we can stimulate a sharper embedding size reduction initially to quickly approach the target sparsity ratio c, and then encourage fine-grained action selection when optimizing towards the balance between performance and space.

2.2.4 Action. At every iteration i, the policy network in the search function G_Φ predicts an action, i.e., dimension $d_n^i \in \mathbb{N}_{>0}$ from interval $[1, d_{max}]$ given a user/item state s_n^i. The action is the embedding size for the corresponding user/item. The recommender $F_\Theta(\cdot)$ takes this action by altering the embedding sizes of users/items (i.e., updating $\mathbf{M}$ in practice), and yields a reward r_n^i along with a subsequent state s_n^{i+1}. The tuple $(s_n^i, d_n^i, r_n^i, s_n^{i+1})$ is defined as a *transition*, which are stored in a replay buffer $\mathcal{B}$ for subsequent training.

2.2.5 Actor and Critic. We adopt an actor-critic paradigm for RL, which provides a better convergence guarantee and lower variance compared with pure policy search methods [10]. Unlike policy-based RL that fails to generalize to continuous action space and cannot extend to unseen actions [7], we adopt a continuous RL backbone, namely TD3 [8]. In CIESS, we have two actor networks $\mu_\mathcal{U}(\cdot)$ and $\mu_\mathcal{V}(\cdot)$ respectively built for the user and item sets. The rationale for having two separate actors is to better accommodate the distributional difference between user and item states, and more concretely, the popularity f_n / f_{max} and recommendation quality indicator q_n. These two actors share the same architecture and only differ in their parameterization. The action/embedding size at the i-th iteration is computed using the actor network given the current user/item state:

$$\hat{d}_u^i = \mu_\mathcal{U}(s_u^i) + \epsilon, \quad u \in \mathcal{U} \quad (11)$$
$$\hat{d}_v^i = \mu_\mathcal{V}(s_v^i) + \epsilon, \quad v \in \mathcal{V}$$

where we add a small noise $\epsilon \sim \mathcal{N}(0, \sigma)$ from Gaussian distribution with standard deviation σ. This is to introduce a small amount of variation to the computed action, hence allowing additional explorations on more possible embedding sizes for the policy.
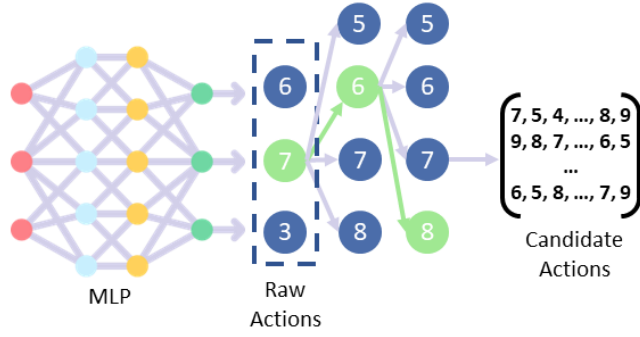
Figure 2: The random walk-based embedding size exploration, where raw actions are $\hat{d}_n^i$ predicted by the actor.

Correspondingly, we build user- and item-specific critic networks to learn to approximate the quality (i.e., Q-value) of an action d_n^i taken at state s_n^i, denoted as $Q_{\mathcal{U}}(s_u^i, d_u^i)$ and $Q_{\mathcal{V}}(s_v^i, d_v^i)$, respectively. Instead of the traditional value-based RL that additionally requires learning a value function [17], the critic networks in the actor-critic framework are trained via tuples stored in the replay buffer $\mathcal{B}$ to map the actor's interaction with the environment to the Q-value, which directs the actors to make policy updates by estimating the quality of the actions taken.

Action Exploration with Random Walk. As discussed in Section 1, such value-based RL optimization heavily relies on learning an accurate critic, which can hardly be guaranteed in continuous embedding size search tasks with numerous actions and states. Although TD3 additionally trains an independent critic network (two additional user/item critic networks in our case) and uses the minimum of their outputs as the predicted Q-value to lower the estimation variance, such a conservative strategy may result in underestimation of the Q-value and eventually suboptimal performance [28]. Therefore, a common practice is to adopt exploration strategies on possible actions. A typical example is the greedy strategy that always selects the action with highest Q-value [7], which can be extremely inefficient to obtain in our large action space. Hence, in addition to the exploration noise in Eq.(11), we further propose a sample-efficient action exploration strategy based on random walk, as shown in Figure 2. To avoid evaluating all possible actions at each state as in the greedy strategy, we only compute Q-values on a sequence of actions mutated from the computed $\hat{d}_n^i$, where the action returning the highest Q-value will be selected. The rationale is, for a given user/item, if we keep fine-tuning its embedding size based on currently the best one $\hat{d}_n^i$ predicted by the actor, there will be a higher chance that we can obtain a more performant embedding size choice. Specifically, to efficiently identify the sequence of actions for exploration, we construct a graph of actions where each action d is connected to a set of neighbor actions $\mathcal{A}_d$. Since each action is an integer-valued embedding size, we can easily measure the distance between any two actions by taking the absolute value of their arithmetic difference. For each action d, the distance to any of its neighbors is below a predefined threshold t:

$$\forall d' \in \mathcal{A}_d : \quad |d' - d| \le t. \tag{12}$$

From each action d in the constructed graph, the probability of reaching action $d' \in \mathcal{A}_d$ during random walk is:

$$p(d \to d') = \frac{|d' - d|}{\sum_{d' \in \mathcal{A}_d} |d' - d|}. \tag{13}$$

Then, starting from $\hat{d}_n^i$, we perform random walk (with replacement) to obtain a series of actions $\mathcal{Z}_{\hat{d}_n^i}$, where we specify $|\mathcal{Z}_{\hat{d}_n^i}|$ to be relatively small to balance efficiency and exploration thoroughness. After obtaining $\mathcal{Z}_{\hat{d}_n^i}$, we evaluate each action in $\mathcal{Z}_{\hat{d}_n^i}$ with the critic network $Q_{\mathcal{U}}(\cdot)/Q_{\mathcal{V}}(\cdot)$ and greedily select the final action d_n^i for iteration i:

$$d_u^i = \mathrm{argmax}_{d' \in \mathcal{Z}_{\hat{d}_u^i}} Q_{\mathcal{U}}(s_u^i, d_u^i), \quad u \in \mathcal{U},$$
$$d_v^i = \mathrm{argmax}_{d' \in \mathcal{Z}_{\hat{d}_v^i}} Q_{\mathcal{V}}(s_v^i, d_v^i), \quad v \in \mathcal{V}, \tag{14}$$

which will be utilized to optimize both the actor and critic networks.

2.3 Selective Retraining for Sparsified Embeddings

We put together a pseudo code for CIESS in Algorithm 1. The embedding size search policy is trained with RL for M episodes, where each episode iterates for T times. In each iteration, CIESS performs random walk-based action exploration and decides the embedding sizes for all users and items (lines 6-9), trains the recommender $F_\Theta(\cdot)$ with the embedding sizes selected to obtain the instant reward r_n^i and next iteration's state s_n^{i+1} (lines 10-11). The transition described in Section 2.2.4 is appended to the replay buffer $\mathcal{B}$ to facilitate optimization of the critic and actor networks in the search function (lines 12-14). We omit the twin network-based optimization process in TD3 and set a pointer to the original paper [8] for finer details. Notably, as a fresh recommender $F_\Theta(\cdot)$ needs to be trained from scratch for every iteration's embedding size decision, we restrict the training process (line 10) to only run for a specified number of epochs (5 in our case) over $\mathcal{D}$ to ensure efficiency. By fixing all hyperparameters and only varying $\mathbf{M}$ for $F_\Theta(\cdot)$, this offers us sufficient confidence in comparing different embedding sizes' performance potential without an excessively time-consuming process. We hereby introduce how to obtain a fully trained recommender under a specific memory budget, which corresponds to line 17 in Algorithm 1.

As stated in Eq.(5), c essentially defines the sparsity of the pruned embedding table $\mathbf{E} \odot \mathbf{M}$, e.g., $c = 0.9$ means only 10% of the parameters in $\mathbf{E}$ are kept. In the RL-based search stage, the policy adjusts the embedding size of each user/item (hence the embedding sparsity) until the reward is maximized. By adjusting λ in the reward, we can let the search function $G_\Phi(\cdot)$ to either emphasis recommendation accuracy or memory efficiency, where the latter is preferred and adopted in this paper as we prioritize high compression rates. As such, $G_\Phi(\cdot)$ will be able to derive a decreasing embedding size for each user/item in the pursuit of maximum reward. However, recall that when obtaining the reward w.r.t. the embedding mask $\mathbf{M}$ computed in each iteration, CIESS does not update the base recommender $F_\Theta(\cdot)$ until full convergence. To make the final decision more robust, we do not rely on a single embedding size decision, and instead maintain a set of candidate embedding mask matrices $\mathcal{M}_c$ with the top-l highest performance measured by q_n during the

---

Algorithm 1 CIESS

1: Initialize the RL-based search function $G_\Phi(\cdot)$;
2: Initialize replay buffer $\mathcal{B}$;
3: **for** $episode = 1, \cdots, M$ **do**
4:     Compute initial state s_n^0 w.r.t. d_{max} for $n \in \mathcal{U} \cup \mathcal{V}$;
5:     **for** $i = 1, \cdots, T$ **do**
6:         /* Each iteration applies to all $n \in \mathcal{U} \cup \mathcal{V}$ */
7:         Initialize base recommender $F_\Theta(\cdot)$;
8:         Perform random walk from $\hat{d}_n^i \leftarrow$ Eq.(11);
9:         Obtain $d_n^i \leftarrow$ Eq.(14) and update $\mathbf{M} \leftarrow$ Eq.(2);
10:        Update $F_\Theta(\cdot)$ w.r.t. Eq.(5) and $\mathbf{E} \odot \mathbf{M}$;
11:        Evaluate $F_\Theta(\cdot)$ to obtain $r_n^i \leftarrow$ Eq.(10), $s_n^{i+1} \leftarrow$ Eq.(6);
12:        Update buffer $\mathcal{B} \leftarrow \mathcal{B} \cup (s_n^i, d_n^i, r_n^i, s_n^{i+1})$;
13:        Draw a batch of transitions from $\mathcal{B}$;
14:        Update $Q_\mathcal{U}(\cdot), Q_\mathcal{V}(\cdot), \mu_\mathcal{U}(\cdot), \mu_\mathcal{V}(\cdot)$ with TD3 [8];
15:     **end for**
16: **end for**
17: Perform selective retraining and obtain $\mathbf{M}^* \leftarrow$ Eq.(16).

---

search stage, constrained by c:

$$\forall \mathbf{M} \in \mathcal{M}_c: \quad 1 - \frac{\|\mathbf{M}\|_{1,1}}{(|\mathcal{U}| + |\mathcal{V}|) \times d_{max}} \geq c \quad (15)$$

Then, in the parameter retraining stage, we retrain the randomly initialized recommender model $F_\Theta(\cdot)$ for each $\mathbf{M} \in \mathcal{M}_c$ till convergence. Afterwards, we select matrix $\mathbf{M}^*$ that yields the highest recommendation quality as the final solution:

$$\mathbf{M}^* = \text{argmax}_{\mathbf{M} \in \mathcal{M}_c} \, \overline{q}, \quad (16)$$

where $\overline{q}$ denotes the mean of all q_n for $n \in \mathcal{U} \cup \mathcal{V}$. If a lower target sparsity $c' < c$ is needed, we can further expand this selective retraining scheme to l best-performing masks $\mathcal{M}_{c'}$ w.r.t. c', thus finding the optimal embedding sizes for different memory budgets in one shot.

3 EXPERIMENTS

We detail our experimental analysis on the performance of CIESS in this section.

3.1 Base Recommenders and Comparative Methods

CIESS can be paired with various base recommender models that utilize embedding-based representation learning. To thoroughly validate our method's versatility and generalizability across different base recommenders, we leverage three widely used recommenders to serve as the base recommender $F_\Theta(\cdot)$, namely neural collaborative filtering (NCF) [14], neural graph collaborative filtering (NGCF) [49], and light graph convolution network (LightGCN) [13], where we inherit the optimal settings reported in their original work and only substitute the embedding table into a sparsified one.

We compete against the following embedding size search algorithms, which are all model-agnostic:

- PEP [27]: It learns soft pruning thresholds with a reparameterization trick to achieve sparsified embeddings.

- ESAPN [26]: It is an RL-based method that automatically searches the embedding size for each user and item from a discrete action space.
- OptEmbed [30]: It trains a supernet to learn field-wise thresholds, and then uses evolutionary search to derive the optimal embedding size for each field.
- Equal Sizes (ES): Its embedding sizes are equal across all users/items and remain fixed.
- Mixed and Random (MR): Its embedding sizes are sampled from a uniform distribution and remain fixed.

3.2 Evaluation Protocols

We perform evaluation on two popular benchmarks, namely MovieLens-1M [12] with 1,000,208 interactions between 6,040 users and 3,952 movies, and Yelp2018 [49] with 1,561,147 interactions between 31,668 users and 38,048 businesses. We split both datasets for training, validation and test with the ratio of 50%, 25%, and 25%.

For effectiveness, we adopt Recall@k and NDCG@k as our metrics by setting $k \in \{5, 20\}$. For CIESS, PEP, ES, and MR, we test the recommendation performance under three sparsity ratios $c \in \{80\%, 90\%, 95\%\}$. For each of these four methods, it is guaranteed that the compressed embedding table has no more than $cd_{max} \times (|\mathcal{U}| + |\mathcal{V}|)$ usable parameters, where the full embedding size $d_{max} = 128$. Notably, since ESAPN and OptEmbed have a more performance-oriented design and do not offer a mechanism to precisely control the resulted embedding sparsity, we only report the performance achieved by their final embedding tables.

3.3 Implementation Notes for CIESS

The subsection details our implementation of the proposed model. Both the base recommender and the search function are trained with Adam optimizer [19]. We train CIESS for a total of $M = 30$ episodes, and each episode contains $T = 10$ iterations. The action space is $[1, 128]$ for the actor network, and the standard deviation $\sigma = 6$ in the Guassian noise. For the random walk component, we set both the threshold t and walk length to 5.

3.4 Comparison of Overall Performance

Table 1 shows the performance of all lightweight embedding methods when paired with different base recommenders. In general, at each specified sparsity $c \in \{80\%, 90\%, 95\%\}$, CIESS significantly outperforms all the precise baselines (i.e., PEP, ES, and MR that have control over the resulted sparsity rate) when paired with all three base recommenders. Specifically, though using a small, fixed embedding size in ES yields competitive recommendation performance compared with PEP's pruning strategy, the embeddings' expressiveness is largely constrained for some important users/items.

Meanwhile, ESAPN and OptEmbed have respectively resulted in a 75% and 79% sparsity rate on average, and failed to meet the lowest 80% target most of the time. Although both have retained relatively more parameters than all other methods in many test cases, their recommendation performance constantly falls behind CIESS. For example, when paired with LightGCN on MovieLens-1M dataset, ESAPN needs to consume more parameters (72% embedding sparsity) to obtain a competitive Recall@5 score w.r.t. CIESS

	LightGCN				NGCF				NCF			
Method	Sparsity	R@5	R@20	N@5	N@20	Sparsity	R@5	R@20	N@5	N@20	Sparsity	R@5	R@20	N@5	N@20
ESAPN	72%	0.0912	0.2422	0.4771	0.4178	85%	0.0856	0.2276	0.4285	0.3829	72%	0.0845	0.2283	0.4454	0.3822
OptEmbed	83%	0.0745	0.1994	0.4257	0.3650	80%	0.0717	0.2038	0.4045	0.3622	85%	0.0458	0.1352	0.2960	0.2573
PEP		0.0771	0.2098	0.4346	0.3778		**0.0806**	0.2138	0.4382	0.3798		0.0725	0.2045	0.4054	0.3603
ES	80%	0.0825	0.2248	0.4536	0.3969	80%	0.0803	**0.2232**	0.4281	**0.3858**	80%	0.0779	0.2080	0.4121	0.3682
MR		0.0737	0.2004	0.4211	0.3648		0.0748	0.2053	0.4202	0.3693		0.0692	0.1914	0.3787	0.3374
CIESS		**0.0920**	**0.2436**	**0.4854**	**0.4257**		0.0800	0.2148	**0.4442**	0.3854		**0.0792**	**0.2221**	**0.4233**	**0.3828**
PEP		0.0733	0.1988	0.4237	0.3641		**0.0794**	**0.2092**	0.4309	0.3727		0.0723	0.2033	0.4053	0.3591
ES	90%	0.0778	0.2105	0.4376	0.3787	90%	0.0747	0.2069	0.4149	0.3679	90%	0.0694	0.1973	0.4011	0.3546
MR		0.0663	0.1802	0.3948	0.3383		0.0745	0.2056	0.4122	0.3660		0.0605	0.1703	0.3523	0.3127
CIESS		**0.0846**	**0.2248**	**0.4631**	**0.4023**		0.0782	0.2082	**0.4385**	**0.3779**		**0.0759**	**0.2131**	**0.4216**	**0.3750**
PEP		0.0659	0.1789	0.3876	0.3315		0.0762	0.2020	0.4248	0.3618		0.0697	0.1955	0.3864	0.3379
ES	95%	0.0646	0.1752	0.3847	0.3272	95%	0.0736	0.1977	0.4134	0.3595	95%	0.0666	0.1841	0.3881	0.3358
MR		0.0594	0.1634	0.3657	0.3119		0.0728	0.1958	0.4152	0.3593		0.0595	0.1637	0.3575	0.3072
CIESS		**0.0744**	**0.2009**	**0.4264**	**0.3664**		**0.0774**	**0.2055**	**0.4333**	**0.3727**		**0.0707**	**0.1982**	**0.4082**	**0.3587**
(a) Results on MovieLens-1M															
ESAPN	76%	0.0257	0.0752	0.0596	0.0699	73%	0.0124	0.0364	0.0232	0.0297	74%	0.0154	0.0448	0.0382	0.0425
OptEmbed	80%	0.0183	0.0534	0.0424	0.0489	67%	0.0133	0.0412	0.0301	0.0369	77%	0.0076	0.0238	0.0161	0.0203
PEP		0.0253	0.0723	0.0605	0.0682		0.0086	0.0275	0.0167	0.0223		0.0130	0.0406	0.0236	0.0327
ES	80%	0.0289	0.0822	0.0665	0.0758	80%	0.0223	0.0665	0.0517	0.0603	80%	0.0131	0.0465	0.0226	0.0351
MR		0.0253	0.0783	0.0588	0.0698		0.0212	0.0618	0.0499	0.0573		0.0101	0.0342	0.0183	0.0268
CIESS		**0.0292**	**0.0839**	**0.0692**	**0.0783**		**0.0233**	**0.0701**	**0.0566**	**0.0652**		**0.0175**	**0.0533**	**0.0377**	**0.0474**
PEP		0.0224	0.0657	0.0531	0.0610		0.0080	0.0276	0.0153	0.0215		0.0139	0.0427	0.0242	0.0335
ES	90%	0.0230	0.0722	0.0544	0.0648	90%	0.0205	0.0622	0.0528	0.0592	90%	0.0119	0.0416	0.0201	0.0309
MR		0.0210	0.0642	0.0476	0.0573		0.0195	0.0573	0.0449	0.0524		0.0089	0.0309	0.0170	0.0242
CIESS		**0.0263**	**0.0730**	**0.0649**	**0.0705**		**0.0232**	**0.0669**	**0.0551**	**0.0619**		**0.0153**	**0.0500**	**0.0350**	**0.0442**
PEP		0.0199	0.0600	0.0496	0.0561		0.0075	0.0259	0.0143	0.0206		0.0125	0.0399	0.0244	0.0330
ES	95%	0.0217	0.0624	0.0494	0.0571	95%	0.0196	0.0573	0.0496	0.0545	95%	0.0091	0.0303	0.0174	0.0238
MR		0.0195	0.0583	0.0465	0.0540		0.0175	0.0528	0.0415	0.0486		0.0078	0.0275	0.0145	0.0216
CIESS		**0.0230**	**0.0657**	**0.0534**	**0.0640**		**0.0222**	**0.0662**	**0.0540**	**0.0613**		**0.0157**	**0.0483**	**0.0383**	**0.0446**
(b) Results on Yelp2018															

Table 1: Performance of all methods on MovieLens-1M (a) and Yelp2018 (b). R@k and N@k are shorthands for Recall@k and NDCG@k, respectively. We highlight the best results when c is set to 80%, 90%, and 95%.

under 80% sparsity. It is also worth noting that, on Yelp2018 dataset, CIESS under 90% sparsity even outperforms ESAPN and OptEmbed with a much lower sparsity (e.g., 67% for OptEmbed 76% for ESAPN). In short, with the same memory consumption, CIESS delivers stronger performance; and at the same performance level, CIESS is more memory-efficient. Hence, the results showcase the continuous embedding size search in CIESS is more advantageous in preserving recommendation accuracy.

Another interesting finding is that, all methods benefit from a performance increase when paired with a stronger recommender, especially the graph-based NGCF and its improved variant LightGCN. NCF is generally less accurate with sparse embeddings, where one possible cause is its matrix factorization component that directly applies dot product to highly sparse user and item embeddings without any deep layers in between. This provides us with some further implications on the choice of base recommenders in a lightweight embedding paradigm.

3.5 Model Component Analysis

The exploration strategy is crucial for finding optimal embedding sizes in CIESS, where we have proposed a combination of Gaussian noise (Eq.(11)) and random walk on the predicted actions. Thus, a natural question is - how useful our random walk-based exploration is, and will a different choice of noise in Eq.(11) substitute its effect?

Sparsity	Noise	Random Walk	MovieLens-1M		Yelp2018	
			R@20	N@20	R@20	N@20
80%	G	Yes	**0.2437**	**0.4250**	**0.0839**	**0.0783**
	OU	Yes	0.2432	0.4243	0.0765	0.0722
	U	Yes	0.2421	0.4241	0.0763	0.0710
	G	No	0.2130	0.3825	0.0766	0.0705
90%	G	Yes	**0.2248**	**0.4023**	0.0730	**0.0705**
	OU	Yes	0.2212	0.3936	**0.0734**	0.0685
	U	Yes	0.2212	0.3971	0.0734	0.0692
	G	No	0.1995	0.3635	0.0701	0.0644
95%	G	Yes	0.2009	**0.3664**	**0.0657**	**0.0604**
	OU	Yes	**0.2027**	0.3570	0.0610	0.0563
	U	Yes	0.1972	0.3554	0.0600	0.0554
	G	No	0.1890	0.3493	0.0603	0.0551

Table 2: Performance of different CIESS variants. OU, N and U denote noises $\mathcal{N}$ sampled from Ornstein-Uhlenbeck process, Gaussian distribution and uniform distribution, respectively. RW indicates whether random walk is in use.

To answer this question, we first conduct a quantitative study with three variants of CIESS. The first two variants respectively use a uniform distribution (U) and an Ornstein-Uhlenbeck process (OU) [44] to replace the Gaussian noise (G) in Eq.(11), while

(a) Training Curve on MovieLens-1M

(b) Training Curve on Yelp2018
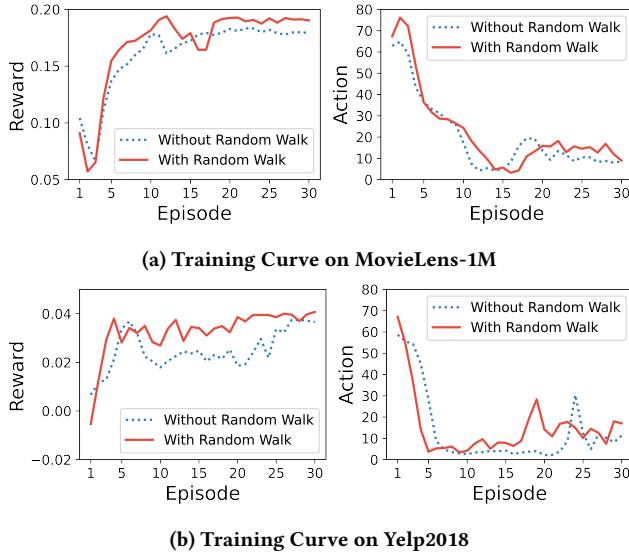
Figure 3: The average reward score and action (embedding size) of CIESS in each training episode on MovieLens-1M (a) and Yelp2018 (b). LightGCN is used as the base recommender.

(a) Effect of λ on MovieLens-1M

(b) Effect of λ on Yelp2018

Figure 4: Sensitivity analysis w.r.t. λ. LightGCN is used as the base recommender.

keeping the random walk component. The third variant retains the Gaussian noise but removes the random walk-based exploration. Table 2 demonstrates the performance change w.r.t. Recall@20 and NDCG@20 in these variants. Due to space limit, we only report results with the best-performing base recommender LightGCN. The results show that, CIESS is relatively insensitive to the choice of noise on the predicted actions, while Gaussian noise obtains better results in most cases. Furthermore, the removal of random walk leads to a significant performance drop, implying the positive contributions from our proposed exploration strategy.

Furthermore, we undertake a qualitative analysis by visualizing and comparing the learning processes of CIESS with and without random walk. Figure 3 displays the average rewards and actions (i.e., embedding sizes) in each episode when both versions are trained. Our observations are: random walk (1) allows the policy to maintain and converge at a higher reward, (2) hit a higher reward sooner, and (3) explores a wider range of actions at both early and final episodes to seek better embedding sizes.

3.6 Hyperparameter Sensitivity Analysis

In this section, we analyze the effect of key hyperparameters in CIESS w.r.t. Recall@20 and NDCG@20. The best base recommender LightGCN is adopted for demonstration.

3.6.1 Reward Coefficient λ. In the reward function Eq.(10), λ balances the accuracy and memory consumption objectives. We tune CIESS with $\lambda \in \{0.2, 0.4, 0.6, 0.8, 1.0, 1.2\}$ for MovieLens-1M and $\lambda \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ for Yelp2018, and show how the ranking quality reacts to the change of λ. As shown in Figure 4(a), CIESS achieves the best performance on MovieLens-1M when $\lambda = 0.4$, and the performance starts to deteriorate when λ is greater than 0.8. When trained on Yelp2018, Figure 4(b) shows that although
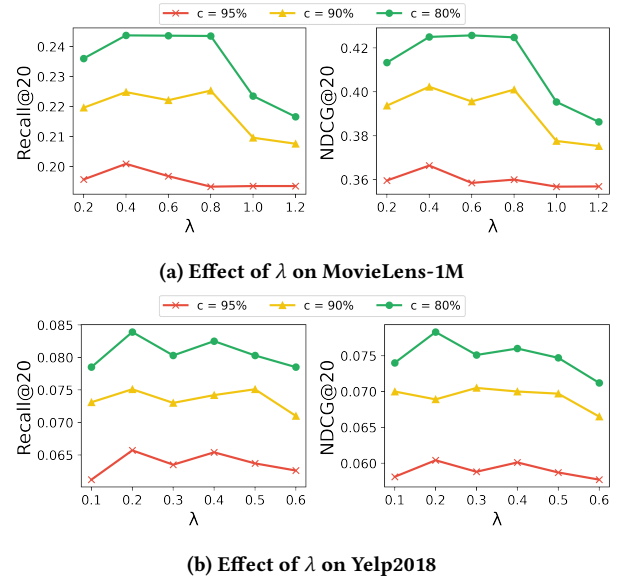
the base model performance peaks when λ is set to 0.2, it is generally insensitive to the value of λ when it ranges between 0.1 and 0.4. After λ reaches 0.5, the recommendation performance starts to decline.

3.6.2 Number of Episodes. To understand the training efficiency of CIESS, we study how many episodes it needs to reach the best performance. So, we examine its recommendation performance throughout the 30 training episodes by segmenting all episodes into 5 consecutive intervals, and each contains 6 episodes. For each interval, we perform selective retraining described in Section 2.3 for all three target sparsity, and report the best performance observed.

Figure 5(a) shows that, when trained on MovieLens-1M, the recommender first reaches its peak performance with $c \in \{80\%, 90\%\}$. The performance of the models within the $c = 90\%$ group reaches its height before the 24th episode, and decreases afterwards. On Yelp2018, Figure 5(b) indicates that the model performance continues to decline in early episodes when the embedding vectors are being excessively compressed. As the policy gradually compensates the performance by allowing bigger embedding sizes for some users/items, the model performance bounces back and then reaches the highest point in the fourth or fifth episode interval. To summarize, setting $M = 30$ is sufficient for CIESS to search for the optimal embedding size when c is high (e.g., 95%), and it will take less time for a lower c (e.g., 80%) to optimize.

4 RELATED WORK

4.1 Latent Factor Recommenders

Neural networks have exhibited superior ability in solving recommendation tasks. Since MLP can learn the non-linear interactions between users and items, feature representation learning with MLP

(a) Effect of M on MovieLens-1M

(b) Effect of M on Yelp2018
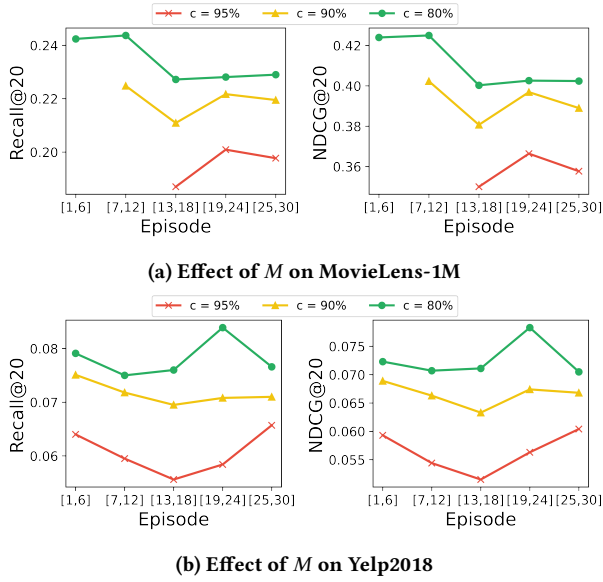
Figure 5: Sensitivity analysis w.r.t. M. LightGCN is used as the base recommender. $c \in \{90\%, 95\%\}$ **is reached only in later episodes on MovieLens-1M.**

has been commonplace. He et al. [14] proposed Neural Collaborative Filtering consisting of an MLP component that learns the non-linear user-item interactions and a generalized matrix factorization component that generalizes the traditional matrix factorization. Cheng et al. [5] proposed Wide&Deep connecting a wide linear model with a deep neural network for the benefits of memorization and generalization. Apart from MLP-based models, graph-based methods [13, 49, 53] have also demonstrated promising capability. Wang et al. [49] proposed the graph neural network-based model NGCF to model the user-item interactions with a bipartite graph and propagate user-item embeddings on the embedding propagation layers. Similarly, He et al. [13] resorted to Graph Convolution Network (GCN) and proposed LightGCN that simplified NGCF by including only its neighborhood aggregation component for collaborative filtering. LightGCN performs neighborhood filtering by computing the weighted sum of the user and item embeddings of each layer. Another promising direction, Factorization Machine [35], along with its deep variants QNFM [3], DeepFM [11] and xLightFM [16] have been studied. [1] was also created to mine higher-order interactions. In addition, heterogeneous data such as textual [2, 9, 22] and visual [21, 48] data has also been exploited by several CNN or RNN-based models. Most of these recommenders utilize vectorized embeddings, which can be optimized by CIESS for memory-efficient embeddings.

4.2 AutoML for Recommendation.

Designing deep recommender models heavily relies on the expertise of professionals. To alleviate the need for human engagement, Automated Machine Learning for Recommender Systems has been studied to automate the design process for recommender models in a task-specific and data-driven manner. So far several research

directions including feature selection search [29, 50], embedding dimension search [26, 27, 32, 58], feature interaction search [41, 42], model architecture search [6, 39], and other components search [31, 57] have been proposed [59]. The first kind reduces computation cost by filtering feature fields based on learnable importance scores [29, 50]. The second line of works proposes dynamic embedding sizes for each feature [26, 27, 32, 58]. The third kind prevents recommender models from enumerating all high-order feature interactions when learning the interactions between features [41, 42]. Model architecture search models explores different network architectures and determines the optimal architecture [6, 39]. Other components search focuses on optimize the loss function and feature interaction function [31, 57].

CIESS falls into the second group and can derive the optimal embedding sizes in a continuous action space.

5 CONCLUSION

Latent factor recommenders use vectorized embeddings with a single and uniform size, leading to substandard performance and excessive memory complexity. To overcome this issue, we proposed an RL-based, model-agnostic embedding size search algorithm, CIESS, that can select tailored embedding size from a continuous interval for each user/item, thus refining the representation expressiveness with minimal memory costs. Future work could explore richer and more robust importance modeling by incorporating signals such as weight magnitudes [15, 40, 43] or model confidence [33, 34]. Such extensions could improve capacity allocation for rare-but-informative entities and mitigate popularity bias.

REFERENCES

[1] Mathieu Blondel, Akinori Fujino, Naonori Ueda, and Masakazu Ishihata. 2016. Higher-Order Factorization Machines. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 3359–3367.

[2] Tong Chen, Hongzhi Yin, Guanhua Ye, Zi Huang, Yang Wang, and Meng Wang. 2020. Try This Instead: Personalized and Interpretable Substitute Recommendation. In *SIGIR*. 891–900.

[3] Tong Chen, Hongzhi Yin, Xiangliang Zhang, Zi Huang, Yang Wang, and Meng Wang. 2021. Quaternion Factorization Machines: A Lightweight Solution to Intricate Feature Interaction Modeling. *IEEE Transactions on Neural Networks and Learning Systems* (2021), 1–14. https://doi.org/10.1109/TNNLS.2021.3118706

[4] Tong Chen, Hongzhi Yin, Yujia Zheng, Zi Huang, Yang Wang, and Meng Wang. 2021. Learning Elastic Embeddings for Customizing On-Device Recommenders. In *SIGKDD*. 138–147.

[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*. 7–10.

[6] Mingyue Cheng, Zhiding Liu, Qi Liu, Shenyang Ge, and Enhong Chen. 2022. Towards Automatic Discovering of Deep Hybrid Network Architecture for Sequential Recommendation. In *WWW*. 1923–1932.

[7] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).

[8] Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80. 1587–1596.

[9] Yuyun Gong and Qi Zhang. 2016. Hashtag recommendation using attention-based convolutional neural network.. In *IJCAI*. 2782–2788.

[10] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, 6 (2012), 1291–1307.

[11] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. *IJCAI* (2017).

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

[13] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*. 639–648.

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*. 173–182.

[15] Steven A. Janowsky. 1989. Pruning versus clipping in neural networks. *Phys. Rev. A* 39 (1989), 6600–6603.

[16] Gangwei Jiang, Hao Wang, Jin Chen, Haoyu Wang, Defu Lian, and Enhong Chen. 2021. XLightFM: Extremely Memory-Efficient Factorization Machine. In *SIGIR*. 337–346.

[17] Manas R Joglekar, Cong Li, Mei Chen, Taibai Xu, Xiaoming Wang, Jay K Adams, Pranav Khaitan, Jiahui Liu, and Quoc V Le. 2020. Neural input search for large scale recommendation models. In *SIGKDD*. 2387–2397.

[18] Wang-Cheng Kang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Ting Chen, Lichan Hong, and Ed H. Chi. 2021. Learning to Embed Categorical Features without Embedding Tables for Recommendation. In *SIGKDD*. 840–850.

[19] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*, Yoshua Bengio and Yann LeCun (Eds.).

[20] Walid Krichene and Steffen Rendle. 2022. On sampled metrics for item recommendation. *Commun. ACM* 65, 7 (2022), 75–83.

[21] Joonseok Lee, Sami Abu-El-Haija, Balakrishnan Varadarajan, and Apostol Natsev. 2018. Collaborative deep metric learning for video understanding. In *Proceedings of the 24th ACM SIGKDD International conference on knowledge discovery & data mining*. 481–490.

[22] Piji Li, Zihao Wang, Zhaochun Ren, Lidong Bing, and Wai Lam. 2017. Neural rating regression with abstractive tips generation for recommendation. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 345–354.

[23] Yang Li, Tong Chen, Peng-Fei Zhang, and Hongzhi Yin. 2021. Lightweight self-attentive sequential recommendation. In *CIKM*. 967–977.

[24] Defu Lian, Haoyu Wang, Zheng Liu, Jianxun Lian, Enhong Chen, and Xing Xie. 2020. Lightrec: A memory and search-efficient recommender system. In *WWW*. 695–705.

[25] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *ICLR*.

[26] Haochen Liu, Xiangyu Zhao, Chong Wang, Xiaobing Liu, and Jiliang Tang. 2020. Automated embedding size search in deep recommender systems. In *SIGIR*. 2307–2316.

[27] Siyi Liu, Chen Gao, Yihong Chen, Depeng Jin, and Yong Li. 2021. Learnable Embedding sizes for Recommender Systems. In *ICLR*.

[28] Zongwei Liu, Yonghong Song, and Yuanlin Zhang. 2023. Actor-Director-Critic: A Novel Deep Reinforcement Learning Framework. *arXiv preprint arXiv:2301.03887* (2023).

[29] Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. 2019. Autocross: Automatic feature crossing for tabular data in real-world applications. In *SIGKDD*. 1936–1945.

[30] Fuyuan Lyu, Xing Tang, Hong Zhu, Huifeng Guo, Yingxue Zhang, Ruiming Tang, and Xue Liu. 2022. OptEmbed: Learning Optimal Embedding Table for Click-through Rate Prediction. In *CIKM*. 1399–1409.

[31] Ze Meng, Jinnian Zhang, Yumeng Li, Jiancheng Li, Tanchao Zhu, and Lifeng Sun. 2021. A general method for automatic discovery of powerful interactions in click-through rate prediction. In *SIGIR*. 1298–1307.

[32] Liang Qu, Yonghong Ye, Ningzhi Tang, Lixin Zhang, Yuhui Shi, and Hongzhi Yin. 2022. Single-Shot Embedding Dimension Search in Recommender System. In *SIGIR*. 513–522.

[33] Yunke Qu, Kevin Roitero, David La Barbera, Damiano Spina, Stefano Mizzaro, and Gianluca Demartini. 2022. Combining Human and Machine Confidence in Truthfulness Assessment. *J. Data and Information Quality* 15, 1 (2022).

[34] Yunke Qu, Kevin Roitero, Stefano Mizzaro, Damiano Spina, and Gianluca Demartini. 2021. Human-in-the-Loop Systems for Truthfulness: A Study of Human and Machine Confidence. In *Conference for Truth and Trust Online*.

[35] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International conference on data mining*. 995–1000.

[36] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. 452–461.

[37] Naser Sedaghati, Te Mu, Louis-Noel Pouchet, Srinivasan Parthasarathy, and P Sadayappan. 2015. Automatic selection of sparse matrix representation on GPUs. In *ACM International Conference on Supercomputing*. 99–108.

[38] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In *SIGKDD*. 165–175.

[39] Qingquan Song, Dehua Cheng, Hanning Zhou, Jiyan Yang, Yuandong Tian, and Xia Hu. 2020. Towards automated neural interaction discovery for click-through rate prediction. In *SIGKDD*. 945–955.

[40] Nikko Ström. 1997. Sparse connection and pruning in large dynamic artificial neural networks. In *Proc. 5th European Conf. on Speech Communication and Technology*. 2807–2810.

[41] Yixin Su, Rui Zhang, Sarah Erfani, and Zhenghua Xu. 2021. Detecting beneficial feature interactions for recommender systems. In *AAAI*. 4357–4365.

[42] Yixin Su, Yunxiang Zhao, Sarah Erfani, Junhao Gan, and Rui Zhang. 2022. Detecting arbitrary order beneficial feature interactions for recommender systems. In *SIGKDD*. 1676–1686.

[43] Georg Thimm and Emile Fiesler. 1995. Evaluating pruning methods. In *Proc. of the Intl. Symposium on Artificial neural networks*. 20–25.

[44] G. E. Uhlenbeck and L. S. Ornstein. 1930. On the Theory of the Brownian Motion. *Physical Review Journals* 36 (1930), 823–841.

[45] Pauli Virtanen et al. 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods* 17, 3 (2020), 261–272.

[46] Qinyong Wang, Hongzhi Yin, Tong Chen, Zi Huang, Hao Wang, Yanchang Zhao, and Nguyen Quoc Viet Hung. 2020. Next Point-of-Interest Recommendation on Resource-Constrained Mobile Devices. In *WWW*. 906–916.

[47] Shoujin Wang, Longbing Cao, Yan Wang, Quan Z Sheng, Mehmet A Orgun, and Defu Lian. 2021. A survey on session-based recommender systems. *ACM Computing Surveys (CSUR)* 54, 7 (2021), 1–38.

[48] Suhang Wang, Yilin Wang, Jiliang Tang, Kai Shu, Suhas Ranganath, and Huan Liu. 2017. What Your Images Reveal: Exploiting Visual Contents for Point-of-Interest Recommendation. In *Proceedings of the 26th International Conference on World Wide Web*. 391–400.

[49] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *SIGIR*. 165–174.

[50] Yejing Wang, Xiangyu Zhao, Tong Xu, and Xian Wu. 2022. Autofield: Automating feature selection in deep recommender systems. In *WWW*. 1977–1986.

[51] Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Guandong Xu, and Quoc Viet Hung Nguyen. 2022. On-Device Next-Item Recommendation with Self-Supervised Knowledge Distillation. In *SIGIR*. 546–555.

[52] Xin Xia, Junliang Yu, Qinyong Wang, Chaoqun Yang, Nguyen Quoc Viet Hung, and Hongzhi Yin. 2023. Efficient On-Device Session-Based Recommendation. *ACM Trans. Inf. Syst.* 41, 4 (2023).

[53] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Lizhen Cui, and Quoc Viet Hung Nguyen. 2022. Are Graph Augmentations Necessary? Simple Graph Contrastive Learning for Recommendation. In *SIGIR*. 1294–1303.

[54] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J. Mankowitz, and Shie Mannor. 2018. Learn What Not to Learn: Action Elimination with Deep Reinforcement Learning. (2018), 3566–3577.

[55] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete Collaborative Filtering. In *SIGIR*. 325–334.

[56] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–38.

[57] Xiangyu Zhao, Haochen Liu, Wenqi Fan, Hui Liu, Jiliang Tang, and Chong Wang. 2021. AutoLoss: Automated Loss Function Search in Recommendations. In *SIGKDD*. 3959–3967.

[58] Xiangyu Zhao, Haochen Liu, Wenqi Fan, Hui Liu, Jiliang Tang, Chong Wang, Ming Chen, Xudong Zheng, Xiaobing Liu, and Xiwang Yang. 2021. Autoemb: Automated embedding dimensionality search in streaming recommendations. In *2021 IEEE International Conference on Data Mining (ICDM)*. 896–905.

[59] Ruiqi Zheng, Liang Qu, Bin Cui, Yuhui Shi, and Hongzhi Yin. 2022. AutoML for Deep Recommender Systems: A Survey. *ACM Transactions on Information Systems* 41, 4 (2022).