A Generic Network Compression Framework for Sequential Recommender Systems

Yang Sun
University of Science and Technology of China
yang.sun@siat.ac.cn

Fajie Yuan*
Tencent
fajieyuan@tencent.com

Min Yang†
SIAT, Chinese Academy of Sciences
min.yang@siat.ac.cn

Guoao Wei
SIAT, Chinese Academy of Sciences
ga.wei@siat.ac.cn

Zhou Zhao
Zhejiang University
zhaozhou@zju.edu.cn

Duo Liu
SIAT, Chinese Academy of Sciences
duo.liu@siat.ac.cn

ABSTRACT

Sequential recommender systems (SRS) have become the key technology in capturing user's dynamic interests and generating high-quality recommendations. Current state-of-the-art sequential recommender models are typically based on a sandwich-structured deep neural network, where one or more middle (hidden) layers are placed between the input embedding layer and output softmax layer. In general, these models require a large number of parameters to obtain optimal performance. Despite the effectiveness, at some point, further increasing model size may be harder for model deployment in resource-constraint devices. To resolve the issues, we propose a compressed sequential recommendation framework, termed as CpRec, where two generic model shrinking techniques are employed. Specifically, we first propose a block-wise adaptive decomposition to approximate the input and softmax matrices by exploiting the fact that items in SRS obey a long-tailed distribution. To reduce the parameters of the middle layers, we introduce three layer-wise parameter sharing schemes. We instantiate CpRec using deep convolutional neural network with dilated kernels given consideration to both recommendation accuracy and efficiency. By the extensive ablation studies, we demonstrate that the proposed CpRec can achieve up to 4~8 times compression rates in real-world SRS datasets. Meanwhile, CpRec is faster during training & inference, and in most cases outperforms its uncompressed counterpart. Our code is available at https://github.com/siat-nlp/CpRec.

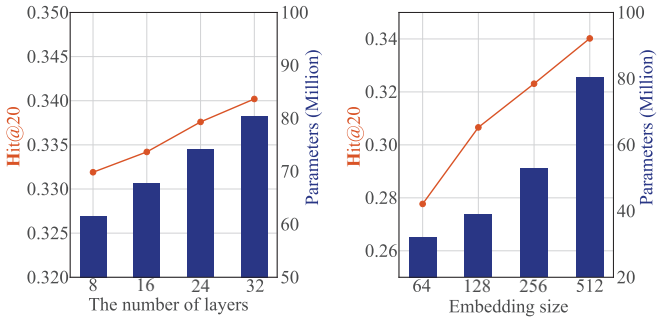
KEYWORDS

Recommender systems, Model compression, Model acceleration

*Joint first author.
†Min Yang is corresponding author. This work was conducted when Yang Sun was interning at SIAT, Chinese Academy of Sciences.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGIR '20, July 25–30, 2020, Virtual Event, China
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8016-4/20/07…$15.00
https://doi.org/10.1145/3397271.3401125

ACM Reference Format:
Yang Sun, Fajie Yuan, Min Yang, Guoao Wei, Zhou Zhao, and Duo Liu. 2020. A Generic Network Compression Framework for Sequential Recommender Systems . In *43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401125



(a) Hit@20 vs. the number of layers **(b) Hit@20 vs. embedding size**

Figure 1: Illustration of prediction accuracy (hit@20) with respect to the model size of NextItNet on ML100 (see Section 4). (a) A deeper model leads to better accuracy where embedding size $d = 512$; (b) A wider embedding leads to highter accuracy where the number of layers r is set to 32.

1 INTRODUCTION

Sequential (a.k.a. session-based) recommender systems (SRS) have become a research hotspot in the recommendation field. This is because user interaction behaviors in real-life scenarios often exist in a form of chronological sequences. In such scenarios, traditional RS based on collaborative filtering [25] or content features [21] fail to model user's dynamic interests and offer only sub-optimal performance. By contrast, sequential recommender models based on recurrent neural networks (RNN) [14, 23] or convolutional neural network (CNN) (often with dilated kernels) [42] have obtained state-of-the-art performance since these models are more powerful in capturing sequential dependencies in the interaction sequence.

In general, modern sequential recommender models based on deep neural networks (DNN) can be split into three major modules:

an embedding layer for representing the interaction sequence, a softmax layer for generating the probability distribution of next item, and one or more hidden (either recurrent or convolutional) layers that are sandwiched between them. To increase the capacity of such models, a larger model size with more parameters is a common method in practice. As shown in Figure 1, the prediction accuracy of the sequential recommender model NextItNet [42] can be largely improved by increasing its model size, i.e., using a larger embedding dimension (denoted by d) or a deeper network architecture (denoted by r). Particularly, NextItNet obtains more than 20% accuracy gains by simply increasing d from 64 to 512, along with about 3 times larger parameters.

Though a large network often brings obvious accuracy improvements, it may also become the major obstacle for model deployment and real-time prediction, especially for memory-limited devices, such as GPU/TPU or end-user devices Another drawback is that both the training and inference speeds will be impacted by employing larger matrices and deeper networks. In addition, many published works also evidence that at some point further increasing model size may cause the overfitting problem [41] or unexpected model degradation [18]. Hence, we argue that model compression is essential to achieve compact recommender models that enable real-time responses and better generalization ability.

In fact, the model size problem in the recommender systems domain is much more challenging than other domains, such as computer vision (CV) [6] and natural language processing (NLP) [1, 5]. For example, in CV, the well-known ResNet-101 [44] trained for ImageNet has only 44.5 million parameters [2]. One of the largest NLP model BERT-Large (24 layers, 16 attention heads) has around 340 million trainable parameters [5]. By contrast, in industrial recommender systems, such as Youtube and Amazon, there exist several hundred million items. If we simply assume the number of items as 100 million and set the embedding dimension d to 1024, we can achieve over 2×100 billion trainable parameters w.r.t. embedding & softmax matrices, which is more than 4000 and 400 times larger than ResNet-101 and BERT-Large, respectively. On the other hand, parameters from the middle layers cannot be ignored in medium-sized or small-scale recommender systems, such as the future in-vehicle recommender systems [22], where memory consumption may be dominant by both middle layers and embedding matrices. Taking NextItNet as an example, the number of parameters in middle layers is $1024 \times 1024 \times 3 \times 32 \approx 100$ million (still much larger than ResNet-101) where 3 and 32 is the kernel size and layers, respectively. Actually, in practice, more convolutional layers may be required for better accuracy if user behavior sequences are longer. Therefore, to reduce the size of sequential recommender models, we need to consider parameter-reduction for both embedding & softmax matrices and middle layers.

To address the aforementioned issues, we present two generic model compression methods to lower memory consumption for SRS. First, to reduce parameters in the embedding and softmax matrices mentioned above, we propose block-wise adaptive decomposition to approximate the original large emebdding[1] matrices. Specifically, we separate all candidate items into clusters according to their frequencies, and the embedding matrix of each cluster, referred to as block, is decomposed by two low-rank matrices, where the rank value is also determined by the item frequencies in the cluster — a larger rank value is assigned to the blocks with more frequent items, and vice versa. Our idea here is motivated by the well-known finding that the item frequency distribution in recommender systems is generally long-tailed — i.e., only a few items may contain rich information due to their high frequency, while others may only contain limited information. Given this structure, a fixed large embedding dimension for all items is redundant and may lead to sub-optimal performance. By the block-wise adaptive decomposition, we are able to assign different dimensions to the block of each cluster. Second, motivated by the cross-layer parameter sharing method of ALBERT [18], we introduce cross-block, adjacent-layer and adjacent-block parameter sharing methods to reduce parameters in the middle layers. Since the two parameter-reduction methods are orthogonal, we can naturally combine them together to achieve a higher compression rate. We name the proposed joint compression framework CpRec.

We summarize our main contributions as follows.

- We propose a block-wise adaptive decomposition method to approximate the original large input/output embedding matrices in SRS. Unlike the standard low-rank decomposition [4], our method enables a better and fine-grained approximation by exploiting the inherent structure of item distribution. To the best of our knowledge, CpRec is the first model compression method in SRS, which directly targets reducing parameter size of the embedding matrices.
- Inspired by the class-based softmax [19] for language model, we design a probability approximation method based on a tree representation in the softmax layer, which replaces the vanilla softmax and notably reduces the training time. Unlike [19], we perform class-based softmax on the decomposed embedding matrix.
- We propose three different layer-wise parameter sharing methods to reduce redundant parameters in the middle layers, which effectively constrains the parameter size as the model grows deeper.
- We obtain a compression ratio of 4~8x on four real-world SRS datasets. Moreover, we demonstrate that CpRec outperforms the uncompressed counterpart in most cases and is faster for both training and generating.

2 RELATED WORK

2.1 DNN-based SRS

Recently, deep neural networks (DNNs) have brought great improvements for SRS and almost dominate this field. Thus far, three types of DNN models have been explored for SRS. Among them, Recurrent Neural Networks (RNNs) are often a natural choice for modeling sequence data [8]. GRU4Rec[14, 29] is regarded as the seminal work that firstly applied gated recurrent units (GRU) architecture for sequential recommendation tasks. Inspired by them, a variety of RNN variants have been proposed to address the sequential recommendation problems, such as personalized SRS with hierarchical RNN [39], content- & context-based SRS [7, 26], data augmentation-based SRS [29]. While effective, these RNN-based models seriously depend on the hidden state of the entire past, which cannot take full advantage of modern parallel processing resources [42], such as GPU/TPU. By contrast, convolutional neural

[1]Without special mention, both the input embedding matrix and the output softmax matrix are unifiedly called an embedding matrix in the following description.

networks (CNNs) and pure attention-based models do not have such limitations since the entire sequence is already available during training. In addition, CNN and attention-based sequential models can perform better than RNN recommenders since much more hidden layers can be stacked by the residual block architecture [11]. To be more specific, [42] proposed a CNN-based generative model called NextItNet, which employs a stack of dilated convolutional layers to increase the receptive field when modeling long-range sequences. Likewise, self-attention based models, such as SASRec [17] and BERT4Rec [28] also obtained competitive results. Compared with NextItNet, the self-attention mechanism is computationally more expensive since calculating self-attention of all timesteps requires quadratic complexity and memory.

All above mentioned sequential recommender models consist of three major modules: two embedding layers for input & output items and several middle layers sandwiched between them. In this paper, we focus on exploring model compression technology for these sandwich-like recommender models. For the below description, we specify CpRec by using the NextItNet architecture, although it can be directly applied to a broad range of recommendation models, such as GRU4Rec and SASRec, etc.

2.2 Model Compression

Training larger and deeper neural networks have become a common practice to obtain state-of-the-art results for a variety of tasks. Meanwhile, the performance gain is usually accompanied by the price of slow processing speed and a huge amount of memory, which makes these models difficult to be deployed on devices with limited resources. As such, more and more attention has been paid to model compression methods. In general, compression methods can be divided into four categories.

Low-rank factorization. The basic idea of this method is to factorize the original large matrix into a product of two low-rank matrices[43]. In terms of model compression, it is mainly used for compressing fully-connected and convolutional layers. Specifically, [4, 44] argued that there was significant redundancy in the parameterization of deep neural networks. As a result, they used two smaller matrices to represent the weight matrix learned from images to reduce free parameters. [24] proposed using low-rank factorization to reduce parameters in the softmax layer. A more recent work in ALBERT [18] performed standard low-rank approximation to decompose both the input and softmax layers into two smaller matrices. By contrast, our proposed method is distinct from them because we perform embedding low-rank factorization adaptively for different blocks according to their item frequencies.

Weight quantization. Network quantization uses low-precision float to represent weight values, which is a general compression method for any deep learning model. For instance, [6, 9, 37] quantized weight and then applied k-means clustering to identify shared weights. [34] used 8-bit quantization to convert activation and intermediate layer weights, which easily leads to significant memory compression. Similarly, [38] proposed single-level quantization method for high-bit quantization and multi-level method for low-bit quantization. Due to its generality, network quantization can also be applied to CpRec for further compression.

Parameter pruning and sharing. Parameter pruning improves parameter efficiency by cutting out redundant parameters. For example, [10, 27] pruned unimportant connections iteratively to lower the storage and computation resources required by the model. ALBERT [18] and [3] adopted the cross-layer parameter sharing method. While model sizes were shrunk notably, the performance of their models also significantly decreased. [1] tie the weights of the input embedding layer and output softmax layer to achieve further compression after the low-rank factorization.

Knowledge Distillation (KD). The basic idea of KD-based compression is to transfer knowledge from a large, pre-trained teacher model to a student one that is typically smaller. Recently, [32] proposed the first KD-based model for the learning to rank problem in recommendation. They showed that the proposed KD method achieved similar ranking performance but with smaller model size and higher inference efficiency. However, throughout the paper, they only investigated the proposed ranking distillation on two very shallow embedding models. The performance of KD on larger & deeper recommender models keeps completely unknown. In addition, they only obtained 2~2.5x compression rate, which is much smaller than that of CpRec.

To the best of our knowledge, model compression techniques have not been well studied in recommender systems. One reason may be that authors in existing literature tended to apply a very small embedding dimension (e.g., $d = 5$ for IRGAN on Movielens [36], $d = 10$ for Fossil on Foursquare [12], $d = 20$ & 30 for N-MF [15] & LambdaFM [40] on Yelp, $d = 50$ for NARM on YOOCHOOSE1 [20]) for research purpose. The other reason is that thus far there seems no existing literature using deep learning models higher than 20 layers for the recommendation task. However, as clearly evidenced in Figure 1, a large NextItNet with $d = 512$ and $r = 32$ indeed performs better on the benchmark dataset.

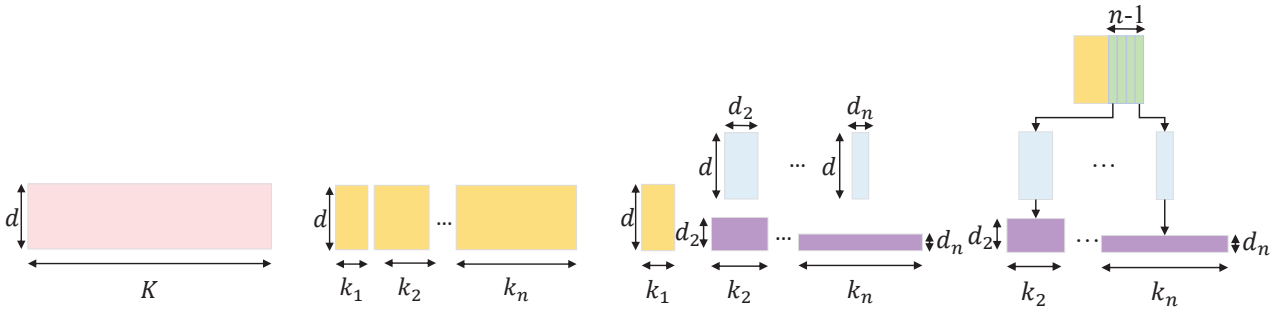
3 METHODS

In this section, we present two main model compression techniques to improve the parameter efficiency of SRS. In what follows, we describe the proposed CpRec by using the NextItNet architecture.

3.1 Block-wise Adaptive Decomposition

In recommender systems, a well-known observation is that frequencies of items generally obey a long-tailed distribution [33, 42, 42], where some "head" (or popular) items have a large number of user interactions, yet only a few interactions are available for the "tail" items. In view of this, we argue that assigning a fixed embedding dimension to all items is sub-optimal and unnecessary. Intuitively, items with higher frequencies may contain more information than the rare ones, and thus should be assigned with more capacity during training. In other words, the embedding dimensions of more frequent (or popular) items are supposed to be larger than those of unpopular items.

An obvious difficulty is that if we set adaptive (i.e.,variable-sized) embeddings to items, they cannot be directly trained by the typical sequential recommender model due to inconsistent dimensions of middle layers. To this end, we perform dimension transformation by multiplying a projection matrix. From a reverse perspective, the transformation process is equivalent to a low-rank factorization



(a) Base (b) Block-partitioning (c) Block-wise adaptive decompostion (for input layer) (d) Block-wise adaptive decompostion (for softmax layer)

Figure 2: Illustration of the proposed block-wise embedding decomposition methods: (a) the vanilla input or output embedding matrix; (b) the embedding matrix of (a) is partitioned into several blocks based on item frequencies; (c) performing low-rank decompostion for blocks in the input layer (d) performing low-rank decompostion for blocks in the output layer based on a two-layer tree structure, where n-1 represents the number of parent classes that the leaf nodes belong to.

given that the original large embedding matrix is reconstructed by two smaller matrices.

More specifically, we first sort all items based on their frequencies $S = \{x_1, x_2, \ldots, x_K\}$, where x_1 and x_K are the most popular and unpopular items, respectively. Denote the number of clusters as n. We partition the item set S into n clusters: $S = S_1 \cup S_2, \ldots, \cup S_n$, $S_1 = \{x_1, x_2, \ldots, x_{k_1}\}$, $S_2 = \{x_{k_1+1}, \ldots, x_{k_1+k_2}\}$, $\ldots, S_n = \{x_{k_1+\ldots+k_{n-1}+1}, \ldots, x_K\}$, where $S_\alpha \cap S_\beta = \emptyset, \alpha \neq \beta$, and the number of items of each cluster is $k_1, k_2, \ldots, k_n$, $\sum_{i=1}^{n} k_i = K$. Accordingly, we can partition the embedding matrix $E \in R^{K \times d}$ in Figure 2a into blocks $E^1 \in \mathbb{R}^{k_1 \times d}, E^2 \in \mathbb{R}^{k_2 \times d}, \ldots, E^n \in \mathbb{R}^{k_n \times d}$ in Figure 2b, where d is the embedding size. Following similar strategy, the softmax matrix $P \in R^{d \times K}$ of output layer can be partitioned into blocks $P^1 \in \mathbb{R}^{d \times k_1}, P^2 \in \mathbb{R}^{d \times k_2}, \ldots, P^n \in \mathbb{R}^{d \times k_n}$. In the following, we will describe block-wise embedding decomposition for the input embedding matrix and output softmax matrix separately.

3.1.1 Adaptive decomposition for the input layer. In the input layer, we factorize the block matrix $E^j, j = 2, \ldots, n$ with two low-rank matrices $E^j = \widehat{E}^j W^j$, where $\widehat{E}^j \in \mathbb{R}^{k_j \times d_j}, W^j \in \mathbb{R}^{d_j \times d}$. d_j is the factorized dimension (a.k.a. rank) for the j-th cluster and we decrease it as the cluster index increases since popular items should have higher expressive capacity. Correspondingly, the embedding representation of each item is different from the one by vanilla look-up operation. Given an item label ID x, we use the following equation to denote its embedding vector $v_x \in \mathbb{R}^d$:

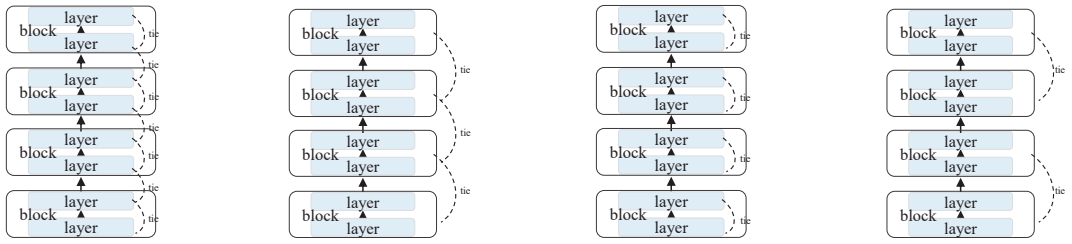
$$v_x = \begin{cases} E^1_x & \text{if } x \in S_1 \\ \widehat{E}^j_g W^j & \text{if } x \in S_j \text{ and } j \neq 1 \end{cases} \quad (1)$$

where $\widehat{E}^j_g$ represents an embedding vector in the g-th row of the j-th block, where $g = x - \sum_{i=1}^{j-1} k_i$. Through this factorization, we reduce the parameters of input layer from $O(K \times d)$ to $O(k_1 \times d + \sum_{i=2}^{n} (k_i + d) \times d_i)$. When $d_i \ll d_1$, the parameters of input layer would be significantly reduced. Figure 2c illustrates the factorization process.

3.1.2 Adaptive decomposition for the softmax layer. It is non-trivial to directly apply the same strategy for the softmax layer since it is unknown which cluster the item belongs to during inference. To address this issue, we explore a straightforward way by simply softmaxing all prediction scores (i.e., logits) of all clusters and recommending top-N items with N highest probabilities. However, we find that this trivial implementation brings a significant decrease on model performance (around 5% to 30% performance drop). We suspect the reason is that it is not accurate to project logits calculated from different clusters to the same softmax space.

Inspired by class-based softmax [19], we structure these blocks by a two-layer tree, where each tree node represents a single cluster. The paradigm of block-wise embedding for the softmax layer is shown in Figure 2d. The embedding matrix of the first cluster (i.e., the first node with dimension d) is saved to the root node, and other blocks are saved to the leaf nodes in the second layer. For the first cluster, each item is represented by one distinct class; whereas for other clusters, we assign two nodes to each item: a root node using its cluster position as its parent class and a leaf node representing the specific position in this cluster. By doing so, the items in the same cluster share the same parent class. To be more specific, we use the similar clustering configuration of block-wise embedding in the input layer. A major difference here is that the first block matrix is extended to $\widehat{P}^1 \in R^{d \times (k_1 + n - 1)}$ in the output layer, where $n - 1$ represents the number of parent classes that the leaf nodes belong to. The label set of the first cluster is extended to $S'_1 = \{1, 2, \ldots, k_1 + n - 1\}$, where $k_1 + 1$ to $k_1 + n - 1$ correspond to the parent class labels of the clusters 2 to n. The other block matrices in the output layer is $P^j = \widehat{W}^j \widehat{P}^j$, where $j \in \{2, \ldots, n\}$, $\widehat{P}^j \in \mathbb{R}^{d_j \times k_j}, \widehat{W}^j \in \mathbb{R}^{d \times d_j}$. Compared with the vanilla softmax layer, we reduce the number of parameters of the output layer from $O(K \times d)$ to $O((k_1 + n - 1) \times d + \sum_{i=2}^{n} (k_i + d) \times d_i)$. In the rest of this section, we describe how to formulate the objective function during training and how to perform generating during inference in detail.

During training, to predict the next item given a context vector $h \in \mathbb{R}^d$ (i.e., the final hidden vector of a sequential recommender model), we need to first determine the search space based on the label of next item, e.g., x. If x belongs to the first cluster, we only compute its logits in this cluster. If x belongs to the other clusters,



(a) Cross-layer (b) Cross-block (c) Adjacent-layer (d) Adjacent-block

Figure 3: Illustration of the cross-layer/block parameter sharing and our proposed adjacent-layer/block parameter sharing.

then we compute the logit in both its parent class and the current cluster. The logit $\hat{y}$ is given as

$$\hat{y} = \begin{cases} h\widehat{P}^1 & \text{if } x \in S_1 \text{ or } \exists c(x) \\ h\widehat{W}^j\widehat{P}^j & \text{if } x \in S_j \text{ and } j \neq 1 \end{cases} \tag{2}$$

where we stipulate that each item x belonging to the leaf node have a parent class label c(x) in the first cluster. Correspondingly, the training process includes two steps. In the first step, the logits of the first cluster are computed, which takes $O(k_1 + n - 1)$ time. In the second stage, if the item label x belongs to one of the leaf class, we compute the logits of that leaf cluster, which takes $O(k_j)$ time. By doing so, we reduce the training time from $O(K)$ using vanilla softmax to between $O(k_1 + n - 1)$ and $O(k_1 + k_j + n - 1)$ using the block-wise embedding.

Let $\hat{p}$ be the normalized value of $\hat{y}$ by the softmax function. The loss function f with respect to $\hat{p}$ and ground-truth label vector y is given

$$f(y, \hat{p}) = \begin{cases} -\sum_{i=1}^{k_1+n-1} y_i \log \hat{p}_i & \text{if } x \in S_1 \\ -\sum_{i=1}^{k_1+n-1} y_i \log \hat{p}_i - \sum_{i=1}^{k_j} y_i \log \hat{p}_i & \text{if } x \in S_j \text{ and } j \neq 1 \end{cases} \tag{3}$$

Different from the training phase, it is unknown which cluster the item belongs to during inference. Yet, we are able to calculate the probability distributions of items in all clusters according to the condition distribution, given as follows:

$$p(x) = \begin{cases} p(x|h, S_1) & \text{if } x \in S_1 \\ p(x|c(x), h)\, p(c(x)|h, S_1) & \text{if } x \in S_j \text{ and } j \neq 1 \end{cases} \tag{4}$$

where $p(x|h, S_1)$, $p(c(x)|h, S_1)$ and $p(x|c(x), h)$ can all be calculated by Eq.(2). Finally, we are able to recommend the top-N items based on $p(x)$. In practice, it is usually not necessary to compute the softmax probabilities for all items during the inference phase. We can perform an early-stop search to speed up the generating process. Specifically, if the top-N probability scores are in the first cluster, we do not need to compute scores in the other clusters (i.e., $p(x|c(x), h)$) since $p(x|c(x), h)\, p(c(x)|h, S_1)$ (where $p(c(x)|h, S_1) < 1$) is always smaller than the top-N scores of the first cluster.

3.2 Layer-wise parameter sharing

User behavior sequence can be very long in many real-world recommender systems, such as short video and news recommendation. To model long-range interaction sequence, a common approach is to build a deeper network architecture. As explained in the introduction section, parameter size in middle layers may dominate the overall memory consumption, especially for small-scale applications in mobile or end-user devices. Hence, the proposed compression method in this section is mainly concentrated on sequential recommender models which have a number of middle layers and every two of them are connected by a residual block (one the most popular ResNet [11] structure), such as NextItNet.

To lower parameter consumption in middle layers, ALBERT [18] proposed the cross-layer parameter sharing approach. While a large number of redundant parameters are reduced, the expressivity of neural network models is also restricted to some extent. In fact, we notice that the performance significantly decreased in both the original paper and our recommendation task by using the cross-layer sharing scheme. To further evidence this observation, we propose an advanced version, namely the cross-block parameter sharing as shown in Figure 3b, where all higher layers reuse parameters of the first residual block (i.e., the two bottom layers).

In order to fully utilize the advantage of stacking layers and meanwhile improve parameter efficiency, we propose another two layer-wise parameter sharing methods: adjacent-layer and adjacent-block parameter sharing. Specifically, the adjacent-layer parameter sharing denotes that the two individual layers in each residual block share the same set of parameters, as shown in Figure 3c. The adjacent-block parameter sharing denotes that parameters are shared between each adjacent residual block, as shown in Figure 3d. Our parameter sharing strategies are considered to bring two major benefits: (1) as a way of regularization, they stabilize the training process and improves model's generalization; (2) they can significantly reduce the number of parameters without degrading performance like the cross-layer parameter sharing. Particularly, we find that the recommendation quality is always slightly better than the base model by the adjacent-block parameter sharing.

3.3 General Architecture of CpRec

By instantiating CpRec using the NextItNet architecture, we show the overal neural network architecture in Figure 4.

3.3.1 Input layer. Given a user-item interaction sequence $\{x_1, x_2, \ldots, x_{t+1}\}$, the recommender model retrieves the embeddings of the first t items $\{x_1, x_2, \ldots, x_t\}$ via look-up table based on the block-wise embedding. After the dimensional projection, we can stack these item embeddings into a new matrix (as shown in the left part of Figure 4, where $t = 5$), which serves as inputs for the middle layers.

3.3.2 Middle layers. As shown in the middle part of Figure 4, we use the dilated convolutional layers [42] for CpRec, where every

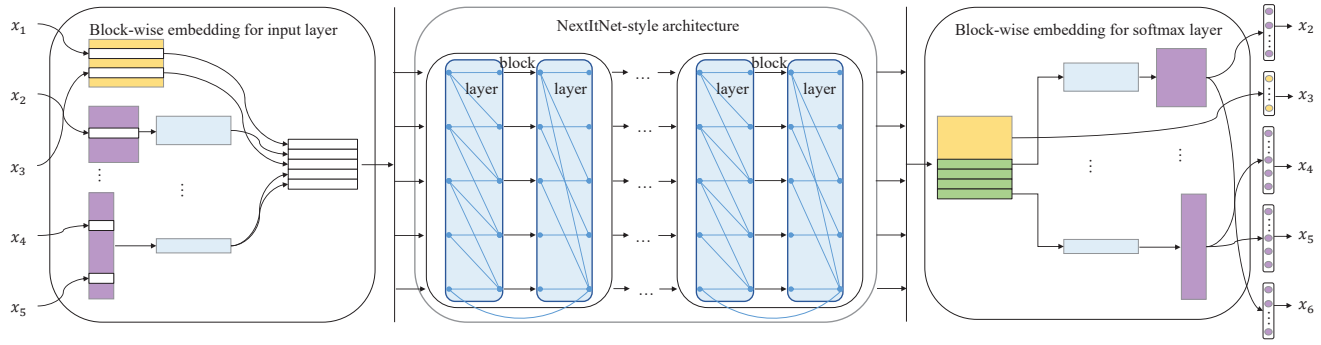


Figure 4: The neural architecture of CpRec.

Table 1: Statistic of the evaluated datasets. "M" and "K" is short for million and kilo, "t" is the length of interaction sequences. For ColdRec, the left and right values devided by '/' denote the source and target dataset, respectively.

Dataset	#items	#actions	#sequences	t
Weishi	66K	10M	1048575	10
ML20	54K	27.7M	1491478	20
TikTok	514K	38.2M	1018155	50
ML100	54K	27.7M	457350	100
ColdRec	191K/21K	82.5M/3.8M	1649095/3798114	50/1~3

two layers are wrapped by a residual block structure. CpRec obtains an exponential growth in the size of the receptive field by doubling the dilation for every layer, e.g., $\{1, 2, 4, 8\}$. In addition, it is a common approach to repeat this structure several times to further enhance model expressiveness and improve accuracy, e.g., $\{1, 2, 4, 8, \ldots, 1, 2, 4, 8\}$. Then, we can apply the proposed layer-wise parameter sharing scheme on these middle layers to improve their parameter efficiency.

3.3.3 Softmax layer. The softmax layer adopts the block-wise embedding decomposition with a tree structure so as to represent blocks of each cluster. As described before, for both the training and inference phases, CpRec can achieve significant speed-up by such a structure. Following NextItNet, given each input sequence $\{x_1, x_2, ..., x_t\}$, CpRec estimates the probability distribution of the output sequence representing $\{x_2, x_3, ..., x_{t+1}\}$, where x_{t+1} is the next item expected.

4 EXPERIMENTAL SETUP

This section introduces our experimental setup including datasets, baseline models, our implementation details and evaluation metrics.

4.1 Datasets

- **Movielens**[2]: The original dataset contains about 280,000 users, 58,000 videos and 27 million user-item interactions with timesteps. To alleviate the impact of cold users and items, we perform the basic pre-processing by filtering out interactions with less than 5 users and users with less than 10 items. Then, we define the maximum length of the interaction sequence as t, and split sequences that exceed the maximum length into multiple sub-sequences. Sequences shorter than t will be padded with zero in the beginning of the sequence to reach t, following [42]. In this paper, we set t to 20 and 100 as short- and long-range sequences respectively, namely, ML20 and ML100.
- **TikTok**[3]: This dataset is released in ICME 2019 short video understanding challenge. It targets at predicting the next preferred videos for a user according to the historical watching behaviors. The original dataset is very large since it contains a large number of cold users and items. We follow the same procedure as in Movielens to remove cold users and items. The dataset after basic pre-processing better fits our GPU memory and helps speed up our experiments.
- **Weishi**[4]: Weishi is a private short-video recommendation dataset collected by Tencent (China). It containing more than 60,000 videos and we set t to 10.
- **ColdRec**[41]: This dataset is used to investigate the transfer learning task based on user interaction sequence. It contains a source dataset and a target dataset. The source dataset contains a userID and his interaction sequence in the QQ Brower[5] recommender system, whereas the target dataset includes the same users and with less than 3 interactions in another recommender system — Tencent Kandian[6]. ColdRec is used as a cold-user based recommendation task by performing transfer learning on the sequential recommender model (i.e., NextItNet).

Table 1 summarizes the statistics of evaluated datasets after basic pre-processing in this work.

4.2 Baseline model

We compare CpRec with two typical sequential recommender models, namely GRU4Rec [14] & Caser [31] and one state-of-the-art model NextItNet. It needs to be noted that we use train GRU4Rec autoregressively, which has a similar effect as the data augmentation method in [29]. Particularly, we perform extensive ablation studies by comparing with NextItNet since they have similar neural network architecture. All models are trained by using the cross-entropy loss [42].

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

[3]https://www.tiktok.com/en/
[4]https://weishi.qq.com
[5]https://brower.qq.com
[6]https://sdi.3g.qq.com/v/201911102006011550

Table 2: Overall performance comparison, including recommendation accuracy, parameter efficiency (Params) , training time and inference speedup (evaluated by the generation of top-5 items). We omit the Params, Training Time (min) and Inference Speedup for GRU4Rec and Caser since they are not comparable to CpRec. MostPop returns item lists ranked by popularity. CpRec with cross-layer [18], cross-block, adjacent-layer and adjacent-block parameter sharing is referred to CpRec-Cl, CpRec-Cb, CpRec-Al and CpRec-Ab respectively.

Data	Model	MRR@5	MRR@20	HR@5	HR@20	NDCG@5	NDCG@20	Training Time (min)	Inference Speedup	Params
Weishi	MostPop	0.0050	0.0121	0.0187	0.0940	0.0083	0.0294	\	\	\
	GRU4Rec	0.1008	0.1159	0.1654	0.3222	0.1168	0.1610	\	\	\
	Caser	0.0910	0.1047	0.1497	0.2937	0.1055	0.1460	\	\	\
	NextItNet	0.1053	0.1205	0.1721	0.3302	0.1218	0.1665	27	1x	74M
	CpRec-Cl	0.1043	0.1193	0.1705	0.3264	0.1206	0.1646	12	2.8x	9M
	CpRec-Cb	**0.1071**	0.1222	0.1736	0.3304	**0.1236**	0.1678	12	2.8x	9M
	CpRec-Al	0.1056	0.1207	0.1719	0.3295	0.1220	0.1665	12	2.8x	11M
	CpRec-Ab	0.1069	**0.1224**	**0.1742**	**0.3320**	**0.1236**	**0.1681**	12	2.8x	11M
ML20	MostPop	0.0044	0.0076	0.0134	0.0485	0.0067	0.0163	\	\	\
	GRU4Rec	0.0946	0.1095	0.1583	0.3122	0.1104	0.1539	\	\	\
	Caser	0.0922	0.1051	0.1511	0.2841	0.1068	0.1444	\	\	\
	NextItNet	0.1057	0.1212	0.1755	0.3352	0.1230	0.1682	104	1x	62M
	CpRec-Cl	0.1011	0.1162	0.1689	0.3243	0.1179	0.1619	64	2.4x	13M
	CpRec-Cb	0.1049	0.1202	0.1747	0.3329	0.1221	0.1669	64	2.4x	14M
	CpRec-Al	0.1040	0.1192	0.1729	0.3309	0.1210	0.1657	64	2.4x	15M
	CpRec-Ab	**0.1063**	**0.1218**	**0.1765**	**0.3364**	**0.1236**	**0.1690**	64	2.4x	15M
TikTok	MostPop	0.0006	0.0015	0.0017	0.0103	0.0009	0.0034	\	\	\
	GRU4Rec	0.0110	0.0154	0.023	0.0708	0.0140	0.0273	\	\	\
	Caser	0.0074	0.0098	0.0146	0.0409	0.0091	0.0164	\	\	\
	NextItNet	0.0120	0.0164	0.0242	0.0715	0.0150	0.0282	300	1x	539M
	CpRec-Cl	0.0130	0.0173	0.0255	0.0732	0.0161	0.0292	64	9.4x	66M
	CpRec-Cb	**0.0133**	**0.0174**	**0.0269**	**0.0755**	**0.0166**	**0.0297**	64	9.4x	67M
	CpRec-Al	0.0130	0.0172	0.0262	0.0741	0.0163	0.0289	64	9.4x	72M
	CpRec-Ab	0.0129	**0.0174**	0.0254	0.0744	0.0158	0.0295	64	9.4x	72M
ML100	MostPop	0.0040	0.0068	0.0124	0.0433	0.0061	0.0146	\	\	\
	GRU4Rec	0.0986	0.1131	0.1622	0.3123	0.1144	0.1568	\	\	\
	Caser	0.0941	0.1072	0.1549	0.2902	0.1091	0.1474	\	\	\
	NextItNet	0.1090	0.1247	0.1781	0.3402	0.1261	0.1720	489	1x	80M
	CpRec-Cl	0.1020	0.1169	0.1674	0.3194	0.1184	0.1618	400	1.5x	13M
	CpRec-Cb	0.1070	0.1221	0.1761	0.3325	0.1241	0.1683	400	1.5x	13M
	CpRec-Al	0.1076	0.1227	0.1760	0.3335	0.1248	0.1690	400	1.5x	25M
	CpRec-Ab	**0.1111**	**0.1265**	**0.1819**	**0.3405**	**0.1286**	**0.1736**	400	1.5x	25M

4.3 Implementation details

We train all models using Tensorflow and Adam optimizer with a learning rate 1×10^{-3} on GPU (Tesla P100). For all shared hyper-parameters, CpRec uses exactly the same as NextItNet for comparison. Regularization and dropout are not applied[7] following the official implementation of NextItNet[8]. Specifically, on Weishi, we use a batch size (denoted by b) of 128 and dilation factors (denoted by l) $2 \times \{1, 2, 2, 4\}$ (8 layers or 4 residual blocks). On ML20 and ML100, we set b to 128, and l to $2 \times \{1, 2, 4, 8\}$ (8 layers) and $8 \times \{1, 2, 4, 8\}$ (32 layers), respectively. On TikTok, we set b to 32 and l to $4 \times \{1, 2, 4, 8\}$ (16 layers). The embedding size d on all above datasets is set to 512. For GRU4Rec and Caser, we use the same embedding and hidden dimensions. Other specific hyper-parameters are empirically tuned according to the results on the testing set. For the transfer learning task on ColdRec, we set b to 64 and 512 on the pre-trained and fine-tuned models, respectively. l and d are set to $4 \times \{1, 2, 4, 8\}$ and 256, respectively. The hyper-parameter settings on ColdRec strictly follow [41].

[7]While we observe that CpRec and NextItNet may perform further better by carefully tuning the regularization in our later work, all conclusions made in this paper hold fully consistent with their regularization variants.
[8]https://github.com/fajieyuan/nextitnet

For CpRec, the model-specific hyper-parameters are the cluster number n, partition rules, and the embedding size of each created block. Empirically, n is very easy to be tuned. For example, If we set n to 2, then we can partition S into two clusters and guarantee that $k_1 : (K - k_1) \approx 2 : 8$ due to the 20/80 principle [16] of the long-tailed distribution. This usually performs well and one can also tune the ratio of $k_1 : (K - k_1)$ greedily for a further better result. If we set n to 3, we first partition it into two clusters and then partition the second cluster and guarantee that $k_2 : (K - k_1 - k_2) \approx 2 : 8$. Following this way, one can obtain the maximum compression ratio by fine tuning n. In this paper, we set n to 5 on TikTok and 3 for the remaining datasets for evaluation purpose. The embedding size of each block can be set in the form of a geometric progression, such as {512, 256, 128}.

4.4 Evaluation Metrics

In order to evaluate the recommendation accuracy of CpRec, we randomly split all datasets into training (80%) and testing (20%) sets. Following previous works [13, 14], we use the popular top-N metrics, including MRR@N (Mean Reciprocal Rank), HR@N (Hit Ratio) and NDCG@N (Normalized Discounted Cumulative Gain), where N is set to 5 and 20. To evaluate the parameter efficiency, we

Table 3: Performance comparison w.r.t. how to apply the block-wise embedding decomposition. NextItNet that uses block-wise decomposition in the input layer, output layer and both are referred to Bi-NextItNet, Bo-NextItNet and Bio-NextItNet, respectively. B1-NextItNet employs the standard low-rank decomposition (i.e., with only 1 block) in the input and softmax layer inspired by [18]. Note that for clarity only the parameters in the input and output matrices are reported in the Params Column. TT is short for training time (unit: min). The inference speedup is simply omitted due to similar results as in Table 2.

Data	Model	MRR@5	HR@5	TT	Params
Weishi	NextItNet	0.1053	0.1721	27	68M
	B1-NextItNet	0.0959	0.1582	18	17M
	Bi-NextItNet	0.1059	0.1724	25	38M
	Bo-NextItNet	0.1060	0.1729	14	38M
	Bio-NextItNet	**0.1068**	**0.1734**	12	8M
ML20	NextItNet	0.1057	0.1755	104	55M
	B1-NextItNet	0.0958	0.1603	79	14M
	Bi-NextItNet	0.1058	0.1761	102	34M
	Bo-NextItNet	0.1063	**0.1766**	66	34M
	Bio-NextItNet	**0.1064**	**0.1766**	64	12M
TikTok	NextItNet	0.0120	0.0242	300	527M
	B1-NextItNet	0.0109	0.0226	162	132M
	Bi-NextItNet	0.0117	0.0239	278	296M
	Bo-NextItNet	0.0122	0.0249	78	296M
	Bio-NextItNet	**0.0123**	**0.0250**	64	66M
ML100	NextItNet	0.1090	0.1781	489	55M
	B1-NextItNet	0.0917	0.1619	439	14M
	Bi-NextItNet	0.1092	0.1791	487	34M
	Bo-NextItNet	0.1090	0.1780	402	34M
	Bio-NextItNet	**0.1109**	**0.1818**	400	12M

report the total number of parameters (without special mention) of each model by Params. To reflect the training efficiency, we report the training time for each model until convergence, denoted by Training Time (min). The inference speedup compared to the baseline is also reported. Similarly to [17, 42], we only evaluate the prediction accuracy of the last item in each interaction sequence in testing set.

5 EXPERIMENTAL RESULTS

The key contribution of CpRec is to improve the memory efficiency for sequential recommender models based on deep neural networks. In this section, we answer the following research questions:

(1) **RQ1**: Does CpRec significantly reduce the model size of a typical sequential neural network, i.e., NextItNet in this paper? If so, does it perform comparably to NextItNet in terms of recommendation accuracy? Are there other advantages that CpRec has over NextItNet?
(2) **RQ2**: What impacts (effectiveness & efficiency) does the adaptive decomposition and layer-wise parameter sharing methods have on CpRec?
(3) **RQ3**: Is CpRec a generic framework that works well for other sequential recommender models, such as GRU4Rec?
(4) **RQ4**: Since sequential recommender models can also be applied for the pre-training and fine-tuning-based transfer learning task [41], does CpRec work as well as the non-compressed model for such a task?

Table 4: The impact of layer-wise parameter sharing strategies. NextItNet with cross-layer, cross-block, adjacent-layer and adjacent-block parameter sharing is denoted by Cl-NextItNet, Cb-NextItNet, Al-NextItNet, Ab-NextItNet, respectively. Note for clarity only the parameters in the middle layers are shown in the Params Column.

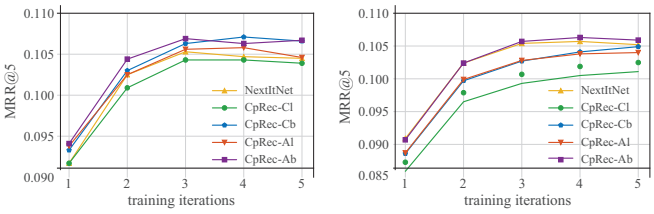
Data	Model	MRR@5	HR@5	Params
Weishi	NextItNet	0.1053	0.1721	6M
	Cl-NextItNet	0.1021	0.1669	1M
	Cb-NextItNet	0.1047	0.1710	2M
	Al-NextItNet	0.1042	0.1699	3M
	Ab-NextItNet	**0.1057**	**0.1727**	3M
ML20	NextItNet	0.1057	0.1755	6M
	Cl-NextItNet	0.1006	0.1678	1M
	Cb-NextItNet	0.1037	0.1728	2M
	Al-NextItNet	0.1037	0.1730	3M
	Ab-NextItNet	**0.1062**	**0.1762**	3M
TikTok	NextItNet	0.0120	0.0242	13M
	Cl-NextItNet	0.0119	0.0241	1M
	Cb-NextItNet	0.0120	0.0243	2M
	Al-NextItNet	0.0118	0.0240	6M
	Ab-NextItNet	**0.0122**	**0.0247**	6M
ML100	NextItNet	0.1090	0.1781	25M
	Cl-NextItNet	0.1007	0.1656	1M
	Cb-NextItNet	0.1048	0.1735	2M
	Al-NextItNet	0.1077	0.1762	13M
	Ab-NextItNet	**0.1101**	**0.1812**	13M

5.1 Quantitative Evaluation (RQ1)

We present the results of CpRec and the baseline models on the four sequential recommendation datasets in Table 2. First, we find that NextItNet performs better than GRU4Rec and Caser with notable improvements in the recommendation accuracy across all datasets. Our observation here is consistent with that in [30, 35, 42].

Second, CpRec with our proposed layer-wise parameter sharing methods (i.e., CpRec-Cb, CpRec-Al & CpRec-Ab) yields competitive results with NextItNet on all evaluation metrics. Particularly, CpRec-Ab performs consistently better than NextItNet — e.g., on TikTok CpRec-Ab obtains 7.5% improvements in terms of MRR@5. Moreover, CpRec has obvious advantages over NextItNet in terms of both parameter efficiency and training/inference time. On the TikTok and Weishi datasets, the compression ratios are up to 7 to 8 times, respectively. By contrast, the state-of-the-art ranking distillation method proposed in [32] achieved only around 2 times the compression ratio. Besides, CpRec is much faster in both training and generating relative to NextItNet, due to the block-wise embedding decomposition and the efficient tree structure in the softmax layer. It is important to note that the reduction of model size does not necessarily lead to a significant improvement in the training speed. Unlike other datasets, the training speed on ML100 is not largely improved by CpRec. This is because on ML100 the time complexity is dominated by the middle layers (32 layers), though the parameters in the middle layers have been significantly reduced.

On the other hand, we observe that CpRec with cross-layer sharing scheme (i.e., CpRec-Cl) yields relatively worse results on ML20 and ML100. As we mentioned before, CpRec-Cl with too few parameters may restrict its model expressiveness. This can be evidenced by CpRec-Cb since it always outperforms CpRec-Cl. Note the improvements of CpRec-Cl over NextItNet on TikTok



(a) Weishi (b) ML20

Figure 5: Convergence behaviors on Weishi and ML20.

Table 5: The effect of adaptive embedding decomposition applied to GRU4Rec. TT is short for training time (unit: min).

Data	Model	MRR@5	HR@5	TT	Params
Weishi	GRU4Rec	0.1008	0.1654	21	69M
	Bi-GRU4Rec	0.1015	0.1666	19	39M
	Bo-GRU4Rec	0.1039	0.1705	7	39M
	Bio-GRU4Rec	**0.1045**	**0.1711**	6	9M
ML100	GRU4Rec	0.0986	0.1622	201	57M
	Bi-GRU4Rec	0.0988	0.1640	199	35M
	Bo-GRU4Rec	0.0995	0.1627	74	35M
	Bio-GRU4Rec	**0.1025**	**0.1683**	72	14M

mainly come from the block-wise embedding decomposition, which is verified by the further ablation studies in Table 3. To show the convergence behaviors of CpRec, we plot the results in Figure 5, which gives similar observations as above.

5.2 Ablation Study (RQ2)

In order to evaluate the impacts of different components on CpRec, we report the ablation test from two aspects: (1) the impact of block-wise adaptive decomposition on the input layer, the output layer, and both. (2) the impact of layer-wise parameter sharing strategies.

5.2.1 Block-wise adaptive decomposition. Table 3 shows the results of block-wise embedding decomposition with 3 different settings as mentioned above. The experimental results clearly demonstrate that Bi- & Bo-based NextItNet achieve similar improvements on parameter efficiency, compressing models to 57% ∼ 74% of the original size. By combining Bi- & Bo-, we obtain Bio-NextItNet, which gives the best compression ratio as well as the highest top-N results and fastest training speed. In addition, we observe that B1-NextItNet performs much worse than NextItNet with around 5∼8% accuracy loss. By contrast, Bio-NextItNet performs better than NextItNet & B1-NextItNet although Bio-NextItNet is more compact than B1-NextItNet. These results suggest that the proposed adaptive decomposition is more effective than the standard decomposition.

5.2.2 Layer-wise parameter sharing. Unlike the block-wise embedding decomposition, layer-wise parameter sharing approaches focus on reducing parameters of the middle layers, and thus are only useful for deep neural network based recommender models. Table 4 presents experimental results for the four layer-wise parameter sharing strategies. Clearly, the results indicate that the proposed adjacent-block strategy (i.e., Ab-NextItNet) always yields the best recommendation accuracy, and even outperforms the standard NextItNet. In addition, it reduces the parameters to half of the original size. Among these approaches, the cross-layer strategy (i.e., Cl-NextItNet) gains the best parameter efficiency, but significantly hurts the model accuracy. By contrast, the cross-block strategy (i.e., Cb-NextItNet) seems also a worth trade-off given its slightly inferior accuracy but a good compression rate. So it may depend on the practical scenario when determining which of the three proposed approaches to use.

Table 6: CpRec vs. NextItNet on the transfer learning task. Note that our evaluation strictly follows [41]. MRR@5 & HR@5 are the finetuned accuracy, whereas Params and training time are evaluated on the pre-trained model, which is computationally more expensive than the finetuned model.

Model	MRR@5	HR@5	Params	Training Time (min)
NextItNet	0.2012	0.3496	102M	578
CpRec-Cb	**0.2035**	**0.3540**	31.5M	118
CpRec-Al	0.2022	0.3507	32.7M	118
CpRec-Ab	0.2025	0.3517	32.7M	118

5.3 Adaptability Experiment (RQ3)

In order to verify the adaptability of CpRec on other sequential recommender models, we specify CpRec with GRU4Rec and report results in Table 5. Similar behaviors have been observed by other datasets, which however are omitted for saving space. Note that since we observe that using more hidden layers does not improve the accuracy of GRU4Rec, we only investigate the block-wise adaptive decomposition for it. Similar to the above experiments, we prefix the model name with Bi-, Bo- and Bio-. As expected, Bio-GRU4Rec obtains the best accuracy, compression rate and training speed. The results can well evidence the applicability and generality of our CpRec.

5.4 Transfer Learning Experiment (RQ4)

Deep learning based sequential recommender models can not only recommending items from where they come from, but also work as a knowledge transfer learning tool to improve recommendation quality in other systems. In other words, we first use CpRec as a pre-trained model and fully train it using the source dataset of ColdRec. Then, we simply add a new softmax layer on the final hidden layer of CpRec, and finetune all parameters on the target dataset by using the pre-trained weights as a warm start. Our transfer learning framework strictly follows a recent work in [41]. We report the results in Table 6.

As shown, CpRec obtains more than 3 times compression rate on ColdRec, but consistently outperforms NextItNet with all proposed layer-wise parameter sharing methods. The HR@5 result of NextItNet reported here is exactly the same as that in the original paper (i.e., FineAll in Table 2 in [41]). In addition, the pretraining time of CpRec is also several times faster than NextItNet, similar to the next item recommendation task.

6 CONCLUSIONS

In this paper, we have proposed CpRec, a flexible & generic neural network compression framework for learning compact sequential recommender models. CpRec significantly reduces parameter size in both the input and softmax layer by leveraging the inherent

long-tailed item distribution. Moreover, CpRec performs further compression by a series of layer-wise parameter sharing methods. Through extensive experiments on real-world datasets, we show that CpRec generates recommendations with higher speed, lower memory and often better accuracy. An important conclusion made from these results is that the commonly used recommender models are not compact at all. Hence, we expect CpRec to be valuable for existing SRS based on deep neural networks.

REFERENCES

[1] Alexei Baevski and Michael Auli. 2018. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853* (2018).

[2] Alexandre Boulch. 2017. Sharesnet: reducing residual network parameter number by sharing weights. *arXiv preprint arXiv:1702.08782* (2017).

[3] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2018. Universal Transformers. *arXiv e-prints*, Article arXiv:1807.03819 (Jul 2018), arXiv:1807.03819 pages. arXiv:cs.CL/1807.03819

[4] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. 2013. Predicting parameters in deep learning. In *Advances in neural information processing systems*. 2148–2156.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[6] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115* (2014).

[7] Youyang Gu, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2016. Learning to refine text based recommendations. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2103–2108.

[8] Guibing Guo, Shichang Ouyang, Xiaodong He, Fajie Yuan, and Xiaohua Liu. 2019. Dynamic item block and prediction enhancing block for sequential recommendation. In *Proc. Int. Joint Conf. Artif. Intell.(IJCAI)*.

[9] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv e-prints*, Article arXiv:1510.00149 (Oct 2015), arXiv:1510.00149 pages. arXiv:cs.CV/1510.00149

[10] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[12] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 191–200.

[13] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. *Proceedings of the 43th International ACM SIGIR conference on Research and Development in Information Retrieval* (2020).

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

[15] Longke Hu, Aixin Sun, and Yong Liu. 2014. Your neighbors affect your ratings: on geographical neighborhood influence to rating prediction. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. ACM, 345–354.

[16] Bin Jiang. 2013. Head/tail breaks: A new classification scheme for data with a heavy-tailed distribution. *The Professional Geographer* 65, 3 (2013), 482–494.

[17] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.

[18] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[19] Hai-Son Le, Ilya Oparin, Alexandre Allauzen, Jean-Luc Gauvain, and François Yvon. 2011. Structured output layer neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 5524–5527.

[20] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 1419–1428.

[21] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. 2011. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*. Springer, 73–105.

[22] Juergen Luettin, Susanne Rothermel, and Mark Andrew. 2019. Future of in-vehicle recommendation systems@ Bosch. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 524–524.

[23] Shilin Qu, Fajie Yuan, Guibing Guo, Liguang Zhang, and Wei Wei. 2020. CmnRec: Sequential Recommendations with Chunk-accelerated Memory Network. *arXiv preprint arXiv:2004.13401* (2020).

[24] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 6655–6659.

[25] Badrul Munir Sarwar, George Karypis, Joseph A Konstan, John Riedl, et al. 2001. Item-based collaborative filtering recommendation algorithms. *Www* 1 (2001), 285–295.

[26] Elena Smirnova and Flavian Vasile. 2017. Contextual sequence modeling for recommendation with recurrent neural networks. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*. ACM, 2–9.

[27] Suraj Srinivas and R Venkatesh Babu. 2015. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149* (2015).

[28] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. *arXiv preprint arXiv:1904.06690* (2019).

[29] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 17–22.

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

[31] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 565–573.

[32] Jiaxi Tang and Ke Wang. 2018. Ranking distillation: Learning compact ranking models with high performance for recommender system. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2289–2298.

[33] Wenting Tu, David W Cheung, Nikos Mamoulis, Min Yang, and Ziyu Lu. 2015. Activity-partner recommendation. In *PAKDD*. 591–604.

[34] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. 2011. Improving the speed of neural networks on CPUs. (2011).

[35] Jingyi Wang, Qiang Liu, Zhaocheng Liu, and Shu Wu. 2019. Towards Accurate and Interpretable Sequential Prediction: A CNN & Attention-Based Feature Extractor. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1703–1712.

[36] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 515–524.

[37] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4820–4828.

[38] Yuhui Xu, Yongzhuang Wang, Aojun Zhou, Weiyao Lin, and Hongkai Xiong. 2018. Deep neural network compression with single and multiple level quantization. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[39] Haochao Ying, Fuzhen Zhuang, Fuzheng Zhang, Yanchi Liu, Guandong Xu, Xing Xie, Hui Xiong, and Jian Wu. 2018. Sequential recommender system based on hierarchical attention networks. In *the 27th International Joint Conference on Artificial Intelligence*.

[40] Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. 2016. Lambdafm: learning optimal ranking with factorization machines using lambda surrogates. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 227–236.

[41] Fajie Yuan, Xiangnan He, Alexandros Karatzoglou, and Liguang Zhang. 2020. Parameter-Efficient Transfer from Sequential Behaviors for User Modeling and Recommendation. *arXiv* (2020), arXiv–2001.

[42] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. 2019. A Simple Convolutional Generative Network for Next Item Recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 582–590.

[43] Fajie Yuan, Xin Xin, Xiangnan He, Guibing Guo, Weinan Zhang, Chua Tat-Seng, and Joemon M Jose. 2018. fBGD: Learning embeddings from positive unlabeled data with BGD. (2018).

[44] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide residual networks. *arXiv preprint arXiv:1605.07146* (2016).