# Learning Multi-granular Quantized Embeddings for Large-Vocab Categorical Features in Recommender Systems

Wang-Cheng Kang*
wckang@ucsd.edu
UC San Diego

Derek Zhiyuan Cheng, Ting Chen, Xinyang Yi,
Dong Lin, Lichan Hong, Ed H. Chi
{zcheng,iamtingchen,xinyang,dongl,lichan,edchi}@google.com
Google

## ABSTRACT

Recommender system models often represent various sparse features like users, items, and categorical features via embeddings. A standard approach is to map each unique feature value to an embedding vector. The size of the produced embedding table grows linearly with the size of the vocabulary. Therefore, a large vocabulary inevitably leads to a gigantic embedding table, creating two severe problems: (i) making model serving intractable in resource-constrained environments; (ii) causing overfitting problems. In this paper, we seek to learn highly compact embeddings for large-vocab sparse features in recommender systems (recsys). First, we show that the novel Differentiable Product Quantization (DPQ) approach can generalize to recsys problems. In addition, to better handle the power-law data distribution commonly seen in recsys, we propose a Multi-Granular Quantized Embeddings (MGQE) technique which learns more compact embeddings for infrequent items. We seek to provide a new angle to improve recommendation performance with compact model sizes. Extensive experiments on three recommendation tasks and two datasets show that we can achieve on par or better performance, with only ~20% of the original model size.

## 1 INTRODUCTION

Representation learning for categorical features has been a very active research area over the last two decades [2, 13, 14]. Although the one-hot encoding is very powerful, learning efficient and effective embeddings for categorical features is challenging, especially when the vocabulary for the sparse features is large, and the training data is highly skewed towards popular items. The size of the embedding table grows linearly with the vocabulary size, leading to two severe problems: (i) making model serving intractable in resource-constrained environments; (ii) causing overfitting problem due to the over-parameterization. This is still the case even for industrial recommender systems with fairly sufficient computing power. For example, in a recent work about YouTube Recommendation [16], it's unveiled that tens of millions of parameters are used to learn embeddings for YouTube video IDs alone.

*work done at Google.


*WWW '20 Companion, April 20–24, 2020, Taipei, Taiwan*

ACM ISBN 978-1-4503-7024-0/20/04.
https://doi.org/10.1145/3366424.3383416

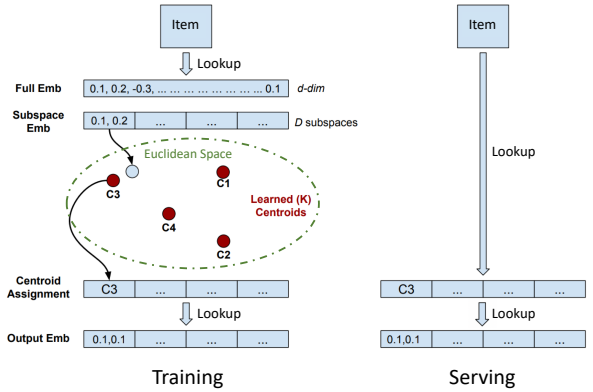



**Figure 1: An illustration of the DPQ embedding lookup process during training and serving. The full embeddings are introduced to aid training, and are discarded during serving.**

To cut down model size from embeddings, there have been some works on embedding compression, such as the hashing tricks [15], low-rank factorization [11], and quantization [6]. On the other hand, to learn better torso and tail item embeddings, there is a line of works seeking to allocate more embedding capacity to frequent words and reduces the capacity for less frequent words with the benefit of reducing overfitting to rare words [1, 4]. Inspired by the two lines of work, we seek to propose a compact embedding scheme with variable capacities. Moreover, many of these efforts were focused on natural language understanding tasks. The embedding compression problem for recommendation tasks remains to be fully studied.

In this paper, we show that the quantization-based embedding compression method Differentiable Product Quantization (DPQ) [6] can generalize to recsys tasks. Moreover, we propose Multi-granular Quantized Embeddings (MGQE) which extend DPQ with variable embedding capacities to adapt to highly skewed data distributions in recsys tasks. MGQE significantly reduces the model size with on par or better performance compared to the full model.

### 1.1 Differentiable Product Quantization

Differentiable Product Quantization (DPQ) [6] is a recent end-to-end embedding compression approach based on a highly compact encoding scheme called KD encoding [5]. Unlike one-hot encoding which maps items to a partially utilized hamming space (i.e., $g : V \rightarrow \{0, 1\}^n$), the KD encoding function is defined as $g : V \rightarrow \{1, \ldots, K\}^D$. For example, an item may be encoded as (1-2-3-1) when $K$=3 and $D$=4. Instead of adopting a pre-defined

assignment (e.g., one-hot encoding) or random mapping (e.g., the hashing trick [15]), the encoding function $g$ in DPQ is differentiable, and can be end-to-end trained with the target task, which allows adaptive assignment of similar codes to semantically similar items. In this paper, we focus on the vector quantization variant of DPQ [6], as we found the softmax based variant does not work well in our preliminary study.

During training, DPQ aims to learn KD codes for items in the vocabulary. The model maintains an embedding vector $\mathbf{e} = [\mathbf{e}^{(1)}; ...; \mathbf{e}^{(D)}] \in \mathbb{R}^d$ for each item, where $D$ is the number of subspaces and $\mathbf{e}^{(i)} \in \mathbb{R}^{d/D}$ is the embedding vector in $i$-th subspace. In the $i$-th subspace, the model also maintains $K$ learnable centroid embeddings $\mathbf{c}_j^{(i)} \in \mathbb{R}^{d/D}$, where $j$=1, ..., $K$. The KD codes are computed through a product quantization process, which contains two processes as follows. In the first encoding process, we find the index of the nearest centroid in each subspace:

$$g(\mathbf{e}) = (\underset{k}{\text{argmin}} \|\mathbf{e}^{(1)} - \mathbf{c}_k^{(1)}\|, ..., \underset{k}{\text{argmin}} \|\mathbf{e}^{(D)} - \mathbf{c}_k^{(D)}\|).$$

And then a decoding function simply retrieves centroid embeddings based on the codes, and concatenates them as the final embedding: $f(k_1, ..., k_D) = [\mathbf{c}_{k_1}^{(1)}; ...; \mathbf{c}_{k_D}^{(D)}]$, where $(k_1, ..., k_D) = g(\mathbf{e})$ are computed KD codes for the item. Although the overall encoding/decoding process $f \circ g$ is not differentiable due to the *argmin* operation. DPQ addresses this issue and makes the process fully differentilable using the straight-through estimator [3].

At serving time, the embedding vector $\mathbf{e}$ is discarded, as we only need to store the codes $(k_1, ..., k_D)$ for each item and the centroid embeddings $\{\mathbf{c}_j^{(i)}\}$. We directly apply the decoding function $f$ on the codes to retrieve the final embedding. Hence, we only need $nD\log(K)$ bits to store code assignments for each item, and $K * D * d/D * 32 = 32Kd$ bits to store the centroid embeddings. Typically, the code assignment is the dominant term as it grows linearly with the vocabulary size $n$. Figure 1 depicts the lookup process in DPQ.

## 2 MULTI-GRANULAR QUANTIZED EMBEDDINGS

With a lower intrinsic dimensionality $D$ and quantized representations, DPQ significantly reduces the model size. However, by assigning same code capacity to each user/item, DPQ is unaware of the highly skewed power-law distributions in typical recsys datasets, where a few popular items dominate the training data and the majority of the items (i.e. long-tail items) are rarely observed. In this case, allocating the same embedding capacity to all items is suboptimal, as it could lead to overfitting on infrequent users/items due to data sparsity and high embedding dimensions. This motivates us to treat frequent and infrequent items differently via learning more compact embeddings for tail items. To this end, we propose Multi-granular Quantized Embeddings (MGQE) which learns embeddings with different capacities for different items. MGQE adopts the quantized embedding DPQ as its underlying embedding scheme for two reasons: (1) DPQ is a highly compact end-to-end embedding learning approach, and achieves excellent compression performance; (2) we found that the quantized encoding scheme (i.e., KD encoding [5]) is highly flexible in terms of supporting different capacities (e.g., varying $D$ and $K$).

### 2.1 Frequency-based Partitions

The first question is how to split items into several groups to which we allocate different embedding capacities. We adopt an intuitive approach that partitions the items based on frequency (i.e., how many times an item appears in the training set). The intuition is that popular items frequently appear in the training set and have more associated observations, and hence we may need a large capacity for them to learn fine-grained embeddings. The frequency-based partition has been recently adopted for NLP models [1, 4], though our work differs in domains (i.e., recommendation models) and underlying embedding approaches (quantized embeddings).

We first assign ascending IDs to the items from the most popular to least popular ones. Then we split the frequency ordered vocabulary $V$ into a multi-tier partition $\widetilde{V} = (V_1, V_2, ..., V_m)$, where $\bigcup_{i=1}^{m} V_i = V$, $V_i \bigcap V_j = \emptyset$ for $i \neq j$, $m$ is the number of groups, $V_1$ contains the most popular items, and $V_m$ contains the least popular items. As we consider recommendation datasets which usually follow power-law distributions, typically we have $|V_1| < |V_2| < ... < |V_m|$. The partition is heuristically set based on dataset statistics, as in [1, 4].

### 2.2 Multi-granular Capacities with Quantized Embeddings

The KD encoding scheme is highly flexible in terms of model sizes, as we can vary the embedding capacity by adjusting $D$ (the number of subspaces) or $K$ (the number of centroids). Hence we can directly derive two variants for the multi-granular capacity allocation via varying K or D. Specifically, instead of using a single $K$ and $D$, we extend DPQ via using a vector to represent the capacity allocation for each group: $\widetilde{K} = [K_1, ..., K_m]$ and $\widetilde{D} = [D_1, ..., D_m]$ where $K_1 \geq K_2 \geq ... \geq K_m$ and $D_1 \geq D_2 \geq ... \geq D_m$. Then we learn DPQ embeddings with $K_i$ and $D_i$ for items in $V_i$. For simplicity, we consider two variants: (1) **variable** $\widetilde{K}$: using fixed number of subspace $D$ with variable $\widetilde{K}$ for each group; (2) **variable** $\widetilde{D}$: using fixed number of centroids $K$ with variable $\widetilde{D}$ for each group. In this way, we allocate multi-granular embedding capacities (and storage space) for items with different popularity.

**A toy example:** Assuming that we have MGQE embeddings with $m = 2, D = 4$, and $\widetilde{K} = [16, 8]$, we will create a DPQ embedding table with $D = 4$ and $K = 16$ for items in $V_1$, and another DPQ embedding table with $D = 4$ and $K = 8$ for items in $V_2$. When retrieving the embedding for an item, we first check which group ($V_1$ or $V_2$) it belongs to, and then lookup in the corresponding DPQ embedding table.

However, a potential drawback of the two variants above is that we need to maintain a private centroid embedding table for each group, which increases the model size ($O(\sum_{i=1}^{m} K_i d)$) for both training and serving. Hence, we propose a multi-granular scheme with centroids shared among groups. That is, we maintain a *single* DPQ embedding table with $D$ and $K$. For items in the $i$-th group, it can only use *first* $K_i$ centroids, instead of all $K$ centroids. In this way, we achieve the multi-granular embedding flexibilities via varying $K$, without additional storage cost. Moreover, we further reduce the model size compared to DPQ, as we only need $D\log(K_i)(i > 1)$ bits to store the code assignment for a tail item (which accounts for majority of the items). We refer to this variant as **shared, variable**

**Algorithm 1** The group-wise embedding look-up process in MGQE.

**Hyper-parameters:** partition $\widetilde{V} = (V_1, V_2, \ldots, V_m)$, embedding dimension $d$, number subspace $D$, variable numbers of centroids $\widetilde{K}$
**Initialization:** intialize a DPQ embedding class with $D$ and $K$
**Input:** a batch of items $S$=($s_1$, $s_2$,…,$s_B$)
Split $S$ into $m$ groups $G_1$, $G_2$, …$G_m$ according to the partition $\widetilde{V}$
**for** $i = 1 \rightarrow m$ **do**
  $\mathbf{E}_i \leftarrow$ MGQE_embedding_lookup[1]$(G_i, K_i)$
$\mathbf{E} \leftarrow$ concatenate($\mathbf{E}_1$, …, $\mathbf{E}_m$)
Reorder $\mathbf{E}$ such that the $i$-th row of $\mathbf{E}$ is the embedding for item $s_i$
**return** $\mathbf{E} \in \mathbb{R}^{B \times d}$

$\widetilde{K}$. By default, we adopt the variant with shared centroids and varying numbers of centroids (**shared, variable** $\widetilde{K}$).

However, as the embedding lookup process (e.g., centroid embedding table to be queried, use the first $K_i$ centroids for items in $V_i$, etc.) is the same for items in the same group, we propose to process the items by groups. Specifically, for a batch of items, we first split them into $m$ groups based on which groups they belong to. Then we obtain the embeddings for each group via $m$ queries of retrieving DPQ embeddings. As the number of groups $m$ is typically small, we found that this implementation of MGQE delivers similar training speed as the vanilla DPQ. Algorithm 1 summarizes the lookup process of MGQE.

# 3 EXPERIMENTS

## 3.1 Datasets

We use two datasets to evaluate both personalized and non-personalized recommendation tasks: **MovieLens-1M**, a widely used benchmark for evaluating collaborative filtering algorithms [7]. The dataset includes 6,040 users and 3,416 items, with a sparsity of 94.44%. As in [8, 10], we treat all ratings as observed implicit feedback instances, and sort the feedback according to timestamps. For each user, we withhold their last two actions, and put them into validation set and test set respectively. All the rest are used for model training. **App-to-app Relevance (AAR)**, an app-to-app relevance dataset collected in Google Play Store to evaluate non-personalized item-to-item recommendation [12]. The dataset includes over 17M app-to-app relevance scores evaluated by human raters. Each pair is unique, and the relevance score ranges from -100 to 100, indicating how relevant a pair of mobile apps are. This relevance dataset is also highly sparse with a sparsity of 99.98%. We randomly split the data into 90% (for training) and 10% (for evaluation). We seek to build a predictive model to estimate matching scores of unseen app pairs.

## 3.2 Backbone Recommendation Models

As quantized embedding is a generic method that can directly replace the embedding layer in existing gradient descent based recommendation models, we include three representative recommendation models as the backbone models to test our hypothesis: **Generalized Matrix Factorization (GMF)** [9] extends the conventional matrix factorization by introducing a learned linear layer for weighting latent dimensions; **Neural Matrix Factorization (NeuMF)** [9] models non-linear interactions between user and item embeddings via multi-layer perceptrons (MLP); **Self-Attentive Sequential Recommendation (SASRec)** [10] is the state-of-the-art method on the sequential recommendation task, which adopts multiple self-attention blocks to capture sequential dynamics in users' action history, and predicts the next item at each time step. The embedding dimensionality $d$ is set to 64 for all methods. Other hyper-parameters are set as suggested in the corresponding papers.

## 3.3 Recommendation Tasks

We conduct our experiments on three representative recommendation tasks: **Task 1: Personalized Item Recommendation**, a conventional task seeks to estimate user-item interactions, and thus can be used to generate personalized recommendations for a user (e.g., "Personalized For You" on your homepage); **Task 2: Sequential Recommendation**, sequential recommendation considers the sequential dynamics in users' action history, and seeks to predict the next item that a user will interact with; **Task 3: Item to item Recommendation**, a non-personalized recommendation task, which seeks to estimate item-item relevance and is often used for recommending related products (e.g., "Related Products" on the product page).

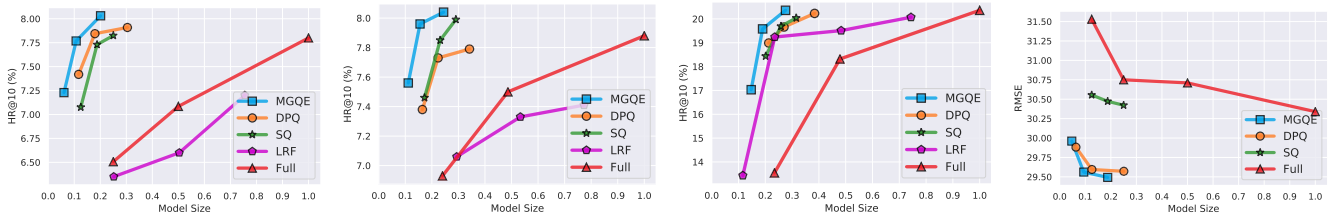
## 3.4 Compression Approaches

To test the effectiveness of our embedding compression technique, we compare them with three baselines: **Full Embedding (FE)**, the conventional approach which learns a full embedding matrix where each row represents the embedding for an item; **Low-rank Factorization (LRF)**. a classic approach to reduce parameters in a matrix. We factorize the embedding matrix into two matrices with size $n \times r$ and $r \times d$; **Scalar Quantization (SQ)**, a classic two-step quantization technique For each dimension in the embedding matrix, SQ records the minimal and maximal values and evenly quantizes the range into $2^b$ buckets, where $b$ is the number of bits; **Differentilable Product Quantization (DPQ)**, DPQ [6] learns subspace centroids and quantizes the embeddings into the nearest centroid. For fair comparison, we implement all the methods using *TemsorFlow*. By default, embedding dimension $d$=64 for all methods, the number of centroids $K$=256 for DPQ. MGQE adopts a two-tier partition where we consider top 10% items as head items and the rest as tail items. The number of centroids $K_1$ is set to 256 for head items, and $K_2$=64 for tail items. To reduce the variance, all the reported numbers are the average of the outcomes from 10 experiments.

## 3.5 Evaluation Metrics

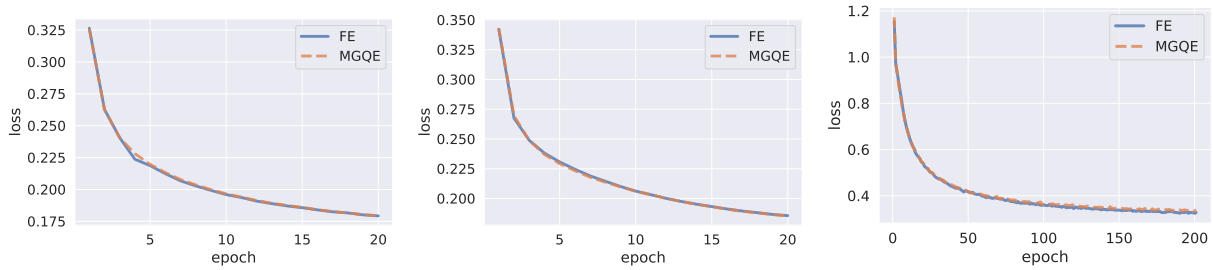
For the personalized recommendation problem, we adopt Hit Rate@K to evaluate recommendation performance [8, 9]. Hit Rate@K counts the fraction of times that the ground-truth next item is among the top K recommended items. Note that since we only have one test item for each user, Hit Rate@10 is equivalent to Recall@10, and is proportional to Precision@10. For the relevance

[1]We extend the embedding look-up procedure in DPQ with an additional parameter $K_i$, which means we only use the first $K_i$ centroids when searching the nearest centroid.




(a) Task 1: MovieLens (GMF) (b) Task 1: MovieLens (NeuMF) (c) Task 2: MovieLens (SASRec) (d) Task 3: AAR (GMF)

**Figure 2: Performance of embedding compression methods with different model sizes.**




(a) Task 1: GMF (b) Task 1: NeuMF (c) Task 2: SASRec

**Figure 3: Training loss of MGQE and full embeddings (FE) on the three recommendations models on Movielens.**

estimation problem on the AAR dataset, we adopt RMSE (Root Mean Square Error) to evaluate model performance.

We evaluate model size using the number of bits needed to store the model during serving. The model size of full embedding is used as the baseline (i.e., 100%), and the reported model sizes of compression approaches are normalized correspondingly.

## 3.6 Results and Discussions on Compression

Figure 2 shows the performance with various compact model sizes on the four tasks. For full embeddings, we vary the dimensionality $d$ to adjust its model sizes. For scalar quantization, we vary the number of bits per dimension. For DPQ/MGQE, we vary the number of subspaces $D$. We can see that the performance of full embeddings drops significantly with smaller model size, which shows directly reducing the dimensionality is not an effective way to compress recommendation models. With the same model size, MGQE has the best performance; and with the same recommendation performance, MGQE has the smallest model size. Hence, MGQE is an effective embedding learning and compression method, and can be applied to achieve better performance and compress recommendation models.

Based on the results of applying embedding compressing methods for three recommendation models (GMF, NeuMF, SASRec) on two datasets (MovieLens and AAR), we found (1) DPQ matches the performance of full embeddings in most cases; (2) MGQE matches (and sometimes improves) the performance with full embeddings in all cases. This verifies that quantized embeddings are able to reduce the model size while matching or even improving the model performance. Moreover, we found MGQE generally outperforms baselines under different compression ratios. This verifies the that MGQE outperforms alternative compression approaches in recsys tasks.

## 3.7 Convergence

As MGQE can directly replace the full embedding layers, it is important to investigate the training of MGQE to check whether the optimization process is the same as or similar to the original one. By doing this, we can check some potential issues like unstable training or slow convergence speed due to the discrete and compact embedding structure. Figure 3 shows the training curves of MGQE on the three recommendation models on the MovieLens dataset, compared with full embeddings. As in our other experiments, we use the default hyper-parameters (e.g., learning rate, batch size, etc.) that are originally designed for full embeddings. We can see that the training processes of MGQE are stable, and closely approximate that of full embeddings. This shows that MGQE behaves similarly to full embeddings in terms of convergence trajectories.

# 4 CONCLUSIONS AND FUTURE WORK

We investigated the embedding compression problem for recommender systems, and proposed multi-granular quantized embeddings (MGQE) for compressing large-vocabulary categorical features. MGQE adopts differentiable quantized representations to reduce the model size, and further cuts down the storage space by using fewer centroids for tail items. MGQE is a generic approach that can be used to replace the embeddings layers in existing recommendation models, and trained end-to-end for the target task. We conducted extensive experiments on compressing three representative recommendation models for three different recommendation tasks. Our results show that MGQE outperforms the baselines, and reaches the full model performance with nearly 20% of the full model size. In the future, we plan to investigate (i) learned fine-grained item partitions for multi-granular capacity allocation; (ii) jointly quantized embeddings for multiple categorical features; and (iii) quantized neural network weights for recommendation models.

# REFERENCES


[1] Alexei Baevski and Michael Auli. 2019. Adaptive Input Representations for Neural Language Modeling. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. https://openreview.net/forum?id=ByxZX20qFQ

[2] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A Neural Probabilistic Language Model. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*. 932–938. http://papers.nips.cc/paper/1839-a-neural-probabilistic-language-model

[3] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *CoRR* abs/1308.3432 (2013). arXiv:1308.3432 http://arxiv.org/abs/1308.3432

[4] Patrick H. Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. 2018. GroupReduce: Block-Wise Low-Rank Approximation for Neural Language Model Shrinking. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*. 11011–11021. http://papers.nips.cc/paper/8295-groupreduce-block-wise-low-rank-approximation-for-neural-language-model-shrinking

[5] Ting Chen, Martin Renqiang Min, and Yizhou Sun. 2018. Learning K-way D-dimensional Discrete Codes for Compact Embedding Representations. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 853–862. http://proceedings.mlr.press/v80/chen18g.html

[6] Ting Chen and Yizhou Sun. 2019. Differentiable Product Quantization for End-to-End Embedding Compression. *CoRR* abs/1908.09756 (2019). arXiv:1908.09756 http://arxiv.org/abs/1908.09756

[7] F. Maxwell Harper and Joseph A. Konstan. 2016. The MovieLens Datasets: History and Context. *TiiS* 5, 4 (2016), 19:1–19:19. https://doi.org/10.1145/2827872

[8] Ruining He, Wang-Cheng Kang, and Julian J. McAuley. 2017. Translation-based Recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys 2017, Como, Italy, August 27-31, 2017*. 161–169. https://doi.org/10.1145/3109859.3109882

[9] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*. 173–182. https://doi.org/10.1145/3038912.3052569

[10] Wang-Cheng Kang and Julian J. McAuley. 2018. Self-Attentive Sequential Recommendation. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*. 197–206. https://doi.org/10.1109/ICDM.2018.00035

[11] 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. *CoRR* abs/1909.11942 (2019). arXiv:1909.11942 http://arxiv.org/abs/1909.11942

[12] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* 7, 1 (2003), 76–80. https://doi.org/10.1109/MIC.2003.1167344

[13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. 3111–3119. http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality

[14] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (1986), 533.

[15] Kilian Q. Weinberger, Anirban Dasgupta, John Langford, Alexander J. Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*. 1113–1120. https://doi.org/10.1145/1553374.1553516

[16] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed H. Chi. 2019. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems, RecSys 2017, Copenhagen, Denmark, September 16-20, 2019*. 269–277. https://doi.org/10.1145/3298689.3346996