HyperRec: Efficient Recommender Systems with Hyperdimensional Computing

Yunhui Guo[1], Mohsen Imani[2], Jaeyoung Kang[1], Sahand Salamat[1], Justin Morris[1], Baris Aksanli[3], Yeseong Kim[4], Tajana Rosing[1]
University of California, San Diego[1] University of California, Irvine[2] San Diego State University[3] DGIST[4]
{yug185,j5kang,sasalama,justinmorris,tajana}@ucsd.edu,m.imanii@uci.edu,baksanli@sdsu.edu,yeseongkim@dgist.ac.kr

ABSTRACT

Recommender systems are important tools for many commercial applications such as online shopping websites. There are several issues that make the recommendation task very challenging in practice. The first is that an efficient and compact representation is needed to represent users, items and relations. The second issue is that the online markets are changing dynamically, it is thus important that the recommendation algorithm is suitable for fast updates and hardware acceleration. In this paper, we propose a new hardware-friendly recommendation algorithm based on *Hyperdimensional Computing*, called HyperRec. Unlike existing solutions which leverages floating-point numbers for the data representation, in HyperRec, users and items are modeled with *binary vectors in a high dimension*. The binary representation enables to perform the reasoning process of the proposed algorithm only using Boolean operations, which is efficient on various computing platforms and suitable for hardware acceleration. In this work, we show how to utilize GPU and FPGA to accelerate the proposed HyperRec. When compared with the state-of-the-art methods for rating prediction, the CPU-based HyperRec implementation is 13.75× faster and consumes 87% less memory, while decreasing the mean squared error (MSE) for the prediction by as much as 31.84%. Our FPGA implementation is on average 67.0× faster and has 6.9× higher energy efficient as compared to CPU. Our GPU implementation further achieves on average 3.1× speedup as compared to FPGA, while providing only 1.2× lower energy efficiency.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

ACM Reference Format:
Yunhui Guo[1], Mohsen Imani[2], Jaeyoung Kang[1], Sahand Salamat[1], Justin Morris[1], Baris Aksanli[3], Yeseong Kim[4], Tajana Rosing[1]. 2021. HyperRec: Efficient Recommender Systems with Hyperdimensional Computing. In *26th Asia and South Pacific Design Automation Conference (ASPDAC '21), January 18–21, 2021, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3394885.3431553

ASPDAC '21, January 18–21, 2021, Tokyo, Japan

ACM ISBN 978-1-4503-7999-1/21/01.
https://doi.org/10.1145/3394885.3431553

1 INTRODUCTION

Many commercial applications such as online shopping websites adopt recommender systems to present products that users will potentially purchase. A fundamental challenge of the recommendation algorithms [1] is to understand and leverage users' preferences for accurate product recommendation by assimilating the large volume of products. Traditional recommender systems [15, 16] typically leverage low-dimensional vectors with floating-point numbers to represent users and items. There are several drawbacks of this approach. First, the user and item information would not be fully exploited due to the low dimensionality of the encoding (embedding) vectors, and it is unclear how to choose a suitable dimensionality. In addition, since there is a significant growth in the number of users and items, the traditional approaches consume a large amount of memory to represent user and item vectors with the full-precision numbers. This representation is thereby hard to scale and also unsuitable for hardware acceleration. Thus, it is important to rethink and develop an efficient representation for recommendation systems in order to achieve fast processing and low resource consumption without compromising prediction quality.

In this paper, we develop the *first* recommendation solution, called HyperRec, which only relies on binary representation and Boolean operations. Although our representation for the users and items is based on *binary*, i.e., lower precision than the existing methods, it can preserve required information in the encoding vectors using a high dimensionality, e.g., $D = 10,000$. We design HyperRec using the principle of Hyperdimensional (HD) Computing [12, 13] which is a brain-inspired computing model. In HyperRec, we encode users, items and ratings using hyperdimensional binary vectors, called *hypervectors*. We represent the relation between users and items via the *binding* and *bundling* operation in HD computing. The recommendation phase is based on the the "nearest neighbor" principle [4] via Hamming distance.

In summary, we show that our proposed HyperRec has the following advantages:

(1) Our evaluation on several large datasets, such as *Amazon*'s datasets, demonstrate that the proposed algorithm is able to improve the recommedataion quality. For example, compared with various previous work on recommender systems, HyperRec decreases the mean squared error (MSE) by as much as 31.84%.
(2) When implementing on CPU, HyperRec is 13.75× faster and reduces the memory consumption by about 87%, as compared with the best performed recommendation algorithm based on SVD++.
(3) Our FPGA implementation is on average 67.0× faster and 6.9× higher energy efficient as compared to CPU. Our GPU

implementation further achieves on average 3.1× speedup as compared to FPGA, while providing only 1.2× lower energy efficiency.

2 RELATED WORK

Recommender systems have drawn much attention from a variety of communities. The emergence of the e-commerce promotes the development of recommendation algorithms. Various approaches have been proposed to provide better product recommendations. Among them, collaborative filtering [6] is a leading technique which recommends the user with products by analyzing similar users' records. We can roughly classify the collaborative filtering algorithms into two categories: neighbor-based methods and latent-factor methods. Neighbor-based methods [1, 26] identify similar users and items for recommendation. Latent-factor models [15, 16] use vector representation to encode users and items, and approximate the rating that a user will give to an item by the inner product of the latent vectors. To give the latent vectors probabilistic interpretations, Gaussian matrix factorization models [21] were proposed to handle extremely large datasets and to deal with cold-start users and items. Neural networks-based recommender systems [2, 9] are proposed recently as generalization of the traditional matrix factorization. Although neural networks have achieved great success in many other areas, recent work [3] pointed out that it is not clear that the neural networks-based recommendation algorithms can really advance the field of recommender systems due to the model complexity. Although such various algorithms have been proposed, to the best of the authors' knowledge, there are no recommendation algorithms that are designed specifically for high efficiency to enable ease of hardware acceleration. There are few research work to accelerate existing algorithms, e.g., an FPGA design for a neighborhood-based collaborative filtering [19]; given the massive amount of data, it is critical to develop new recommender systems in a hardware friendly manner.

3 PRELIMINARY

The proposed algorithm is based on the principle of Hyperdimensional (HD) computing [13]. HD computing [14] is a brain-inspired computing model in which entities are represented as hyperdimensional binary vectors. Hyperdimensional computing has been used in language recognition [10, 11, 22], prediction from multimodal sensor fusion [24, 25], hand gesture recognition [22] and brain–computer interfaces [23]. HD computing is inspired by the understanding that the human brain is more capable of recognizing patterns than calculating with numbers. This fact motivates us to simulate the process of brain's computing with points in high-dimensional space [14]. These points can effectively model the neural activity patterns of the brain's circuits. This capability makes hyperdimensional vectors very helpful in many real-world tasks. The information that contained in hyperdimensional vectors is spread uniformly among all its components in a holistic manner so that no component is more responsible to store any piece of information than another.

We can easily construct a new hypervector based on some old ones using vector or Boolean operations. Such as binding that forms a new hypervector which associates two base hypervectors, and bundling that combines several hypervectors into a single composite hypervector. We introduce several arithmetic operations that are designed for hypervectors.

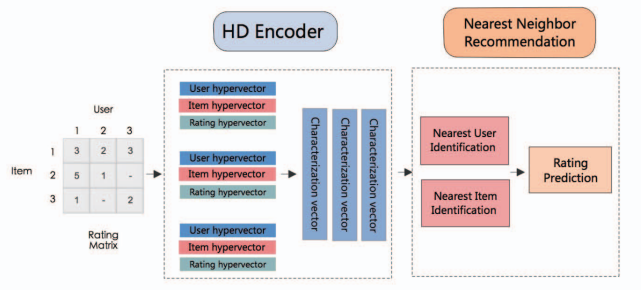

Figure 1: Overview of HyperRec.

- **Component-wise** *XOR*: we can bind two hypervectors A and B by component-wise XOR and denote the operation as $A \otimes B$. The result of this operation is a new hypervector that is dissimilar to its constituents (i.e., $d(A \otimes B; A) \approx D/2$) , where $d()$ is the Hamming distance; hence XOR can be used to associate two hypervectors.
- **Component-wise majority**: bundling operation is done via the component-wise majority function and is denoted as $[A + B + C]$. The majority function is augmented with a method for breaking ties if the number of component hypervectors is even. The result of the majority function is similar to its constituents, i.e., $d([A + B + C]; A) < D/2$. This property makes the majority function well suited for representing sets.

4 HYPERREC

4.1 Overview

In this paper, we propose a new recommendation solution called HyperRec. HyperRec is based on HD computing and leverages the "nearest neighbor" principle for recommendation. Figure 1 shows the overview of HyperRec. HyperRec has two stages to make the recommendation: i) *HD encoding* stage which creates the database from the dataset, and ii) *Recommending product* stage which provides the rating for the products.

In the HD encoding stage, we convert the raw data of users, items and ratings with *hyperdimensional binary vectors*, in short, hypervectors. This is very different from the traditional approaches that represent users and items with low-dimensional full-precision vectors. By representing users and items with hyperdimensional binary vectors, we can save memory as well as enable fast hardware acceleration. Next, we construct the characterization vectors to represent the relation between users and items. Traditional matrix factorization techniques often leverage the latent user and item vectors to approximate the ratings with an iterative optimization procedure [21]. However, it requires the global information of the user-item relation. The proposed relation encoding module only leverages the local information of each user and item which is faster and more scalable. In the next recommending product stage, we make recommendations with the encoded hypervectors based on the "nearest neighbor" principle via Hamming distance. In this section, we will introduce the details of the proposed algorithm.

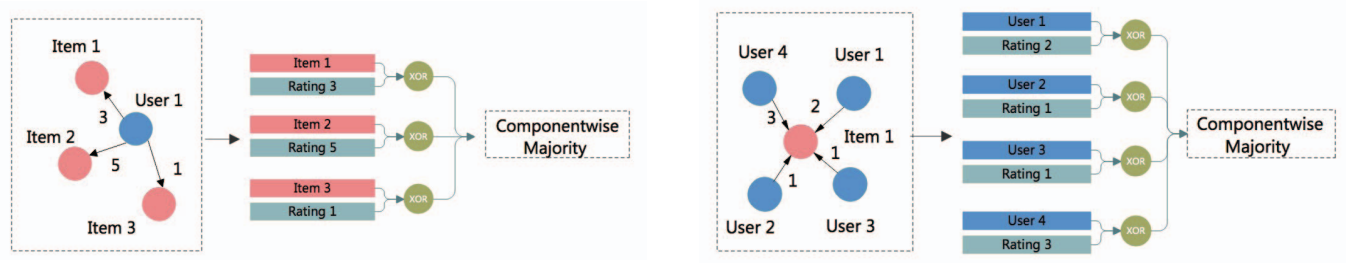

Figure 2: (a): Generation of user characterization vector. (b): Generation of item characterization vector.

4.2 HD Encoding

The encoding strategy of the proposed HyperRec is based on HD computing. We encode all users, items and ratings using hyperdimensional binary vectors. Our goal is to discover and preserve users' and items' information based on their historical interactions in an efficient manner. For each user u and item v, we first randomly generate a D dimensional binary vector, In HD computing, D can be as large as ten thousands. Note that if the binary vector for each rating is generated randomly, the information that consecutive ratings should be similar is lost. Instead, we first generate a hypervector filled with ones for rating 1. To generate the hypervector for rating r, we flip the bits between $(r-2)\frac{D}{R}$ and $(r-1)\frac{D}{R}$ of the hypervector of rating $r-1$ and assign the resulting vector to rating r. By this means, consecutive ratings are close in terms of Hamming distance. Particularly, the Hamming distance between rating r and rating $r-1$ is $\frac{D}{R}$ while the Hamming distance between rating 1 and rating R is D. Following the proposed procedure, we can generate the hypervectors H_r for each rating r.

To encode the relation between users and items, we propose an encoding strategy based on the *binding* and *bundling* operation in HD computing. The intuition of the proposed encoding strategy is that if two users given similar rating to the same item, their should have a high similarity. The proposed encodings can be written as: $C_u = [H_{r_{uv_1}} \otimes H_{v_1} + ... + H_{r_{uv_n}} \otimes H_{v_n}]_{\{v_1,...,v_n \in B_u\}}$ and $C_v = [H_{r_{u_1 v}} \otimes H_{u_1} + ... + H_{r_{u_n v}} \otimes H_{u_n}]_{\{u_1,...,u_n \in B_v\}}$, where $\otimes$ is the XOR operator and $[A + ... + B]$ is the component-wise majority function. r_{uv} is the rating that user u given to item v. By this approach, we can capture the difference between users' consuming behaviors and their rating patterns. For instance, if two users u and u' bought the same item and rated it similarly, the Hamming distance between their characterization hypervectors C_u and $C_{u'}$ will be small. The distance is proportional to the difference of the ratings. Note that we keep the last D/R bits of all rating hypervectors the same, so if two users rated the same item very differently, the Hamming distance between their characterization vectors will still be closer than the users who have no co-purchasing behaviors. The process is shown in Fig. 2.

4.3 Recommending Products

After we obtain the characterization hypervectors of users and items in the form of encoded hypervectors, we use Hamming distance to identify similarity. In order to compute the rating that user u will give to item v, we first identify the k-nearest items of item v based the ratings they received, and denote this set as $N^k(v)$. For each of the k-nearest item $v' \in N^k(v)$, we also identify k'-nearest users of user u in the set $B_{v'}$ based on the ratings they give, and denote this set as $N^{k'}(u, v')$. Then we compute the predicted rating of user u for item v' as $\hat{r}_{uv'} = \mu_u + \frac{\sum_{u' \in N^{k'}(u,v')}(1-dist(u,u'))(r_{u'v'}-\mu_{u'})}{C}$, where C is the normalization factor which is $\sum_{u' \in N^{k'}(u,v')}(1 - dist(u,u'))$. And $dist(u,u')$ is the normalized Hamming distance between the characterization vectors of users u and u'. Then, we compute the predicted rating of user u for item v as $\hat{r}_{uv} = \mu_v + \sum_{v' \in N^k(v)}(1 - dist(v,v'))(\hat{r}_{uv'} - \mu_{v'})$. Similarly, $dist(v,v')$ is the normalized Hamming distance between the characterization vector of item v and item v'. After we obtain the predicted ratings of user u for all the items he/she did not buy before, we can recommend the user u with the items with highest predicted ratings. HyperRec has the simplicity of neighbor-based methods and meanwhile utilizes the effectiveness of HD computing. It is fast since the training phase only needs single pass of the data and the computation consists of only Boolean operations.

5 HYPERREC ACCELERATION

As HyperRec uses binary representation for the encoded data, we can efficiently implement the proposed algorithm on various processors and systems. For example, similar to the existing recommendation solutions, HyperRec can be implemented on the contemporary CPU in a relatively straight-forward way. We can also further accelerate HyperRec on parallel computing platforms, since the operations are either component-wise binary operations and associative search using Hamming distance. In this section, we elaborate our acceleration strategies for two parallel computing platforms, FPGA and GPU.

5.1 HyperRec FPGA Implementation

We use query hypervector to denote a particular user or item characterization hypervector and class hypervector to denote the user or item characterization hypervector used for similarity check. Substantial number of operations that can be executed in parallel in HyperRec is matched with intrinsic parallelism of FPGAs. Each dimension of the encoded hypervector can be generated independent of the other dimensions. The similarity metric between the generated dimensions and the class hypervectors can also be calculated independent of the other dimensions. We fully pipeline the encoding and the associative search of HD such that the encoding module generates D dimensions of the query hypervector and

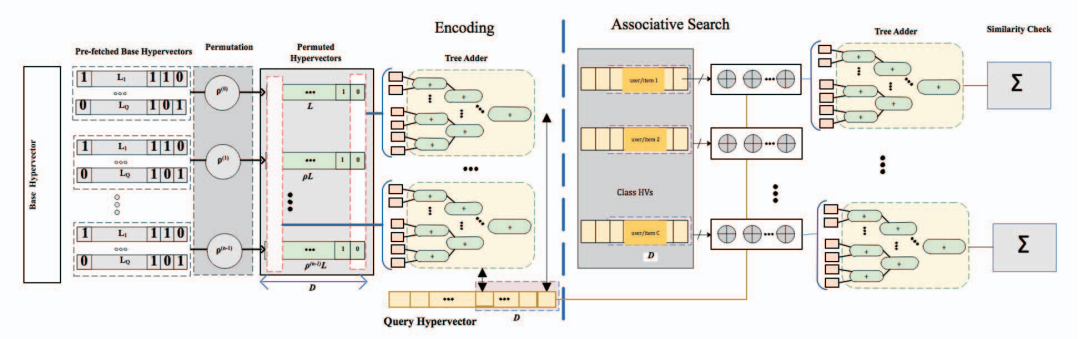

Figure 3: Architecture of HyperRec FPGA-based accelerator.

then the associative search module calculates the similarity metric between these D dimensions and their corresponding dimensions in every class hypervector and accumulates the similarity metric between the query hypervector and each class hypervector.

Figure 3 shows the architecture of HyperRec FPGA-based accelerator. As illustrated in the figure, to generate D dimension of the query hypervector, first we read the base hypervectors for each user or item, permute them and then by using D tree adders, we accumulate the dimensions of all the permuted hypervectors. The generated dimensions of the query hypervector are binarized and passed to the associative search module. D dimensions of the query hypervector are compared against their corresponding D dimensions of every class hypervector using a series of XORs. The outputs of the XORs are aggregated together using C tree adders where C is the number of users or items. HyperRec FPGA-based accelerator accumulates the partial similarity metrics for each user or item. At the end the maximum number between all the similarity metrics represents the output.

5.2 HyperRec GPU Implementation

Before running the GPU program, also called as *kernel*, we first generate hypervectors corresponding to user, item, and rating hypervectors on CPU. Note that this generation is a single-time cost since these hypervectors only depend on the number of users, items, and rating scale but not on the actual data. Next, we copy these hypervectors to GPU memory and encode the characterization vectors. The dimensionality of the hypervectors has a large impact on the performance of HD computing. In the encoding module, threads of a kernel sum up each dimension of the characterization vector. After the accumulation, we normalize and binarize the result hypervector. This *binarization* helps to avoid the divergence and the use of expensive operations. We selectively save the results when the rating of an item and the corresponding user exists. This technique reduces the number of write operations in the global memory of the GPU.

For the recommending products stage, we construct a matrix of pairwise Hamming distance from user and item characterization vector, respectively. The calculation of Hamming distance can simply parallelize over $(user, user)$ or $(item, item)$ pairs. We compute Hamming distance by comparing bit similarity between two characterization vectors. Using the symmetric nature of pairwise Hamming distance (i.e. $dist(u, u') = dist(u', u)$), we only calculate the upper triangular part of the distance matrix which can reduce the global memory access. Even though this adds divergence to each thread, it showed curtailed execution time of pairwise hamming distance calculation.

We parallelize the nearest neighbor computation over users and items, with each thread predicting the rating of a user for a specific item. As mentioned above, this process involves selecting the nearest users and items. Searching similar items can be done before the kernel execution of the nearest neighbor recommendation stage. Once the kernel executes, each thread extracts similar users based on Hamming distance, as finding analogous user depends on the user purchase history. We then select similar top-K users/items using the heap data structure.

6 EVALUATION

6.1 Experimental Setup

We implement the proposed HyperRec on three different platforms, CPU, FPGA, and GPU. The CPU-based HyperRec is implemented on Python 2.7 using Numpy library, which efficiently performs the binary operations using C++ backend. We use the CPU-based implementation to compare with the state-of-the-art algorithms running on CPU. The experiments are run on Intel Core i5 2.9 GHz with 8G RAM. We also evaluate the efficiency of HyperRec on two parallel computing platforms: Xilinx Kintex-7 FPGA, and NVIDIA GPU GTX 1080Ti. For FPGA, we extended the same framework as [27] for implementing HyperRec. For GPU, we used our implementation introduced in Section 5 using CUDA v10.0.

We conduct extensive experiments using several real-world large datasets for evaluating the proposed algorithm as follows:

6.1.1 Datasets. We use datasets from *Movielens* [8], *Amazon* [20], *FilmTrust* [7] and *Yelp* [1]. The *Movielens* datasets contain thousands of ratings that users given to movies over various periods of time. For the *movilens-10M*, we randomly sample 10000 users and for each user we sample 5 items. *Amazon* datasets [20] contain a variety of categories ranging from *Arts* to *Sports*. The *FilmTrust* dataset was crawled from the FilmTrust website in June, 2011 and is a relatively small dataset. *Yelp* dataset contains over 700000 ratings that users given to food and restaurants.

[1] https://www.yelp.com/dataset

Table 1: The mean squared error of all the algorithms on *Movielens, Yelp* and *Filmtrust*.

	KNNBasic	KNNWithMeans	SVD	SVD++	PMF	NMF	SlopeOne	Co Clustering	NCF	GCNN	FM Rec	HyperRec
movielens-100K	0.9811	0.9507	0.9357	0.9215	0.9502	0.9632	0.9442	0.9646	1.3561	0.9236	**0.8923**	0.9649
movielens-1M	0.9335	0.9414	0.8920	0.8797	**0.8734**	0.9190	0.9045	0.9148	1.2120	0.9123	0.9178	0.9195
movielens-10M	1.1597	1.1469	1.0088	1.0043	1.0966	1.1254	1.2100	1.1170	1.2258	**0.9812**	1.1123	1.1030
Yelp	1.1065	1.0832	1.0703	1.0709	1.1550	1.1190	1.1048	1.0822	1.3551	1.2312	1.1123	**1.0604**
Filmtrust	0.9232	0.9203	0.8996	0.8821	1.1562	0.9231	0.9378	0.9416	1.0282	0.9820	0.9210	**0.8806**

Table 2: The mean squared error of all the algorithms on *Amazon*'s datasets.

	KNNBasic	KNNWithMeans	SVD	SVD++	PMF	NMF	SlopeOne	Co Clustering	NCF	GCNN	FM Rec	HyperRec
Arts	1.0380	0.9837	1.0467	1.0109	0.9445	1.1065	0.9897	1.0672	1.2762	1.1231	0.9721	**0.9320**
Tools	1.1370	1.1233	1.0583	1.0437	1.1272	1.2208	1.1457	1.1265	1.2618	1.0123	**0.9981**	1.1506
Sports	0.9874	0.9227	0.9681	0.9423	1.0917	1.0535	0.9311	0.9926	**0.8583**	0.9122	0.9212	0.9108
Musical	1.0784	1.1568	1.0636	1.0593	**1.0551**	1.2772	1.1628	1.1903	1.2092	1.1213	1.2012	1.2965
Clothing	0.7117	0.4768	0.7439	0.6849	0.7262	0.5961	0.4824	0.6362	0.3311	0.3412	0.3348	**0.3250**
Patio	1.2455	1.2536	1.2178	1.2062	**1.1965**	1.3284	1.2716	1.2690	1.5917	1.3324	1.2341	1.2805
Office	1.2241	1.1885	1.1788	**1.1658**	1.2815	1.2587	1.2026	1.2820	1.5197	1.3412	1.3212	1.2527

6.1.2 Baselines for Existing Algorithms. We compare our algorithm with several state-of-the-art rating prediction methods: KNNBasic [6], KNNWithMeans [6], SVD [15], SVD++ [15], PMF [21], NMF [17], SlopeOne [18], Co-clustering [5], NCF [9], GCNN [28] and FM Rec [9]. MSE is used for evaluation since it is the standard metric for comparing rating prediction algorithms [21].

6.2 Recommendation Quality Evaluation

We first evaluate the recommendation quality for different solutions. We randomly select 70% of ratings in each dataset as training dataset and 20% of ratings in each dataset as testing dataset. The rest 10% of ratings are for validation data to tune hyperparameters. We use the best performed hyperparameters on the validation set. For KNNBasic and KNNWithMeans, the number of neighbors is 40. For SVD, SVD++ and PMF, the dimension of the latent factors is 100. For NMF, the dimension of the latent factor is 20. The learning rate of SVD and PMF is 0.005. The learning rate of SVD++ is 0.007. For all the latent-factor based methods, the regularization parameters is 0.02 and the training epoch is 50. For the co-clustering algorithm, we choose the size of the cluster to be 3. For NCF, we follow the recommended parameters settings in the original paper [9]. For HyperRec, the dimension of the hypervectors is set to be 1000. The number of neighboring items is set to be 5 and the number of neighboring users is set to be 30. Please note that prediction results of HyperRec is independent upon the computing platform.

We list the experimental results of all the algorithms in Table 1 and Table 2. The best result on each dataset is shown in bold. As we can see, HyperRec achieves the best results on about half of the considered benchmarks. Compared with neighbor-based methods, our method can capture richer information about users and items with HD computing, which can help us identify similar users and items easily. Compared with latent-factor based methods, HyperRec needs much less memory and is easily scalable. We will show the time and memory comparison of HyperRec with SVD++ in Section 6.3. Note that the neural network based recommendation algorithm, such as NCF, does not achieve competitive performance compared with other methods, which is consistent with the phenomenon observed in a recent work [3]. The possible reason is that neural network based algorithsm can easily overfit the data due to the high complexity of the model.

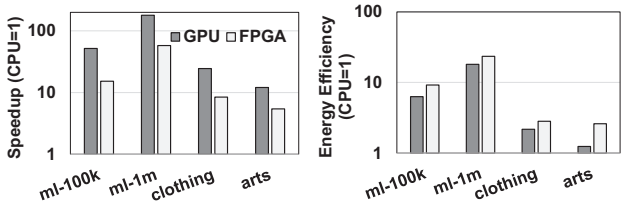

Figure 4: HyperRec Efficiency on GPU and FPGA.

Different from other methods which utilize floating-point numbers, HyperRec stores users and items as hiperdimensional binary vectors rather than full-precision numbers and only relies Boolean operations. These unique properties make it very hardware-friendly so it can be easily accelerated. In the following sections, we will show how the proposed HyperRec is accelerated on various hardware platforms.

Table 3: Efficiency Comparison between SVD++ and HyperRec on CPU.

	Metrics	SVD++	HyperRec
ml-100K	Time	186.08s	25.88s
	Memory	27.56M	2.625M
Arts	Time	3.10s	1.10s
	Memory	23.24M	2.619M
Clothing	Time	43.50s	40.99s
	Memory	102.74M	13.03M
Office	Time	412.2s	9.43s
	Memory	73.32M	7.66M

6.3 Efficiency Evaluation

6.3.1 Comparison with State-of-the-Art Method. We compare the efficiency of HyperRec with SVD++. Since SVD++ is typically run on CPU, we compare the results with the CPU-based HyperRec.

Table 3 compares the memory consumption and execution time of HyperRec with SVD++ on the four datasets of different sizes: *ml-100k*, *Arts*, *Clothing* and *Office*. The results show that HyperRec is about 13.75 times faster than SVD++ on these four datasets which is crucial for real-time applications. In SVD++, users and items are usually represented by one hundred dimension full-precision

vectors. For each user, we need at least 3200 bits to store his/her latent features and 3200 bits to store the gradients that are used for optimization. In contrast, HyperRec need only 1000 bits to represent each user, which amounts to a factor of six of memory saving. Along with the simpler computations of HyperRec, i.e., binary operations, the reduction of the computed data size leads the significant performance improvement. In our measurements, HyperRec reduces the memory consumption by about 87%.

6.3.2 Different Hardware Accelerations. Figure 4 compares HyperRec efficiency on different platforms. All results are reported as compared to energy and execution time of CPU-based HyperRec. Our evaluation shows that FPGA implementation can provide on average 67.0× faster and 6.9× higher energy efficiency as compared to CPU. FPGA exploits lookup-tables to paralleize bitwise operations involves in encoding and Hamming computing modules. GPU further improves FPGA efficiency by (i) providing higher parallelism during similarity search and nearest neighbor modules, and (ii) exploiting data sparsity. By utilizing register on FPGA is not sufficient to handle the sparse data. Hence, frequent access to memory degrades performance. However, GPU are general-purpose and consume more energy than FPGA. Our evaluation shows that our GPU implementation is on average 3.1× faster than FPGA, while providing only 1.2× lower energy efficiency. Note that the efficiency of FPGA and GPU can vary depending on the platforms and devices.

7 CONCLUSION

In this paper, we propose a novel recommendation algorithm called HyperRec. We conducted extensive experiments to show that HyperRec can achieve competitive recommendation quality as compared with the state-of-the-art methods, while improving the performance by 13.75× with much less memory footprints. We also demonstrate that how to accelerate HyperRec on parallel computing platforms. The results show that our FPGA implementation is on average 67.0× faster and 6.9× higher energy efficiency as compared to CPU. Our GPU implementation further achieves on average 3.1× speedup compared to FPGA.

8 ACKNOWLEDGEMENT

This work was supported in part by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA. This work is also supported by NSF CHASE-CI #1730158, NSF FET #1911095, NSF CC* NPEO #1826967. The paper was also funded in part by SRC AIHW grants.

REFERENCES

[1] Charu C Aggarwal. 2016. *Recommender systems*. Springer.
[2] 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.
[3] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 101–109.
[4] Christian Desrosiers and George Karypis. 2011. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook*. Springer, 107–144.
[5] Thomas George and Srujana Merugu. 2005. A scalable collaborative filtering framework based on co-clustering. In *Data Mining, Fifth IEEE international conference on*. IEEE, 4–pp.
[6] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. 1992. Using collaborative filtering to weave an information tapestry. *Commun. ACM* 35, 12 (1992), 61–70.
[7] G. Guo, J. Zhang, and N. Yorke-Smith. 2013. A Novel Bayesian Similarity Measure for Recommender Systems. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*. 2619–2625.
[8] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5, 4 (2016), 19.
[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*. 173–182.
[10] Mohsen Imani, Abbas Rahimi, Deqian Kong, Tajana Rosing, and Jan M Rabaey. 2017. Exploring hyperdimensional associative memory. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 445–456.
[11] Aditya Joshi, Johan T Halseth, and Pentti Kanerva. 2016. Language geometry using random indexing. In *International Symposium on Quantum Interaction*. Springer, 265–274.
[12] Pentti Kanerva. 1988. *Sparse distributed memory*.
[13] Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation* 1, 2 (2009), 139–159.
[14] Pentti Kanerva. 2014. Computing with 10,000-bit words. In *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*. IEEE, 304–310.
[15] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 426–434.
[16] Yehuda Koren and Robert Bell. 2011. Advances in collaborative filtering. In *Recommender systems handbook*. Springer, 145–186.
[17] Daniel D. Lee and H. Sebastian Seung. 2000. Algorithms for Non-negative Matrix Factorization. In *NIPS*.
[18] Daniel Lemire and Anna Maclachlan. 2005. Slope one predictors for online rating-based collaborative filtering. In *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 471–475.
[19] Xiang Ma, Chao Wang, Qi Yu, Xi Li, and Xuehai Zhou. 2015. An FPGA-based accelerator for neighborhood-based collaborative filtering recommendation algorithms. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*. IEEE, 494–495.
[20] Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*. ACM, 165–172.
[21] Andriy Mnih and Ruslan R Salakhutdinov. 2008. Probabilistic matrix factorization. In *Advances in neural information processing systems*. 1257–1264.
[22] Abbas Rahimi, Simone Benatti, Pentti Kanerva, Luca Benini, and Jan M Rabaey. 2016. Hyperdimensional biosignal processing: A case study for EMG-based hand gesture recognition. In *Rebooting Computing (ICRC), IEEE International Conference on*. IEEE, 1–8.
[23] Abbas Rahimi, Pentti Kanerva, José del R Millán, and Jan M Rabaey. 2017. Hyperdimensional computing for noninvasive brain–computer interfaces: Blind and one-shot classification of EEG error-related potentials. In *10th ACM/EAI International Conference on Bio-inspired Information and Communications Technologies (BICT)*.
[24] Okko Rasanen and Sofoklis Kakouros. 2014. Modeling dependencies in multiple parallel data streams with hyperdimensional computing. *IEEE Signal Processing Letters* 21, 7 (2014), 899–903.
[25] Okko J Räsänen and Jukka P Saarinen. 2016. Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns. *IEEE transactions on neural networks and learning systems* 27, 9 (2016), 1878–1889.
[26] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 1–35.
[27] Sahand Salamat, Mohsen Imani, Behnam Khaleghi, and Tajana Rosing. 2019. F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 53–62.
[28] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.