A Fast Sampling Algorithm for Maximum Inner Product Search

Qin Ding
University of California, Davis

Hsiang-Fu Yu
Amazon

Cho-Jui Hsieh
University of California, Los Angeles

Abstract

Maximum Inner Product Search (MIPS) has been recognized as an important operation for the inference phase of many machine learning algorithms, including matrix factorization, multi-class/multi-label prediction and neural networks. In this paper, we propose Sampling-MIPS, which is the first sampling based algorithm that can be applied to the MIPS problem on a set of general vectors with both positive and negative values. Our Sampling-MIPS algorithm is efficient in terms of both time and sample complexity. In particular, by designing a two-step sampling with alias table, Sampling-MIPS only requires constant time to draw a candidate. In addition, we show that the probability of candidate generation in our algorithm is consistent with the true ranking induced by the value of the corresponding inner products, and derive the sample complexity of Sampling-MIPS to obtain the true candidate. Furthermore, the algorithm can be easily extended to large problems with sparse candidate vectors. Experimental results on real and synthetic datasets show that Sampling-MIPS is consistently better than other previous approaches such as LSH-MIPS, PCA-MIPS and Diamond sampling approach.

1 Introduction

Maximum Inner Product Search (MIPS) has been recognized as the main computational bottleneck of the inference phase of many machine learning algorithms, including matrix factorization for recommender systems (Koren et al., 2009), multi-class/multi-label classification (Babbar and Schölkopf, 2017; Yu et al., 2014b), and recurrent neural networks when the output vocabulary size is big (Sutskever et al., 2014). Given a set of d-dimensional candidate vectors $\mathcal{H} = \{\mathbf{h}_1, \ldots, \mathbf{h}_n\}$ and a query vector $\mathbf{w} \in \mathbb{R}^d$, the problem of MIPS is to find the vector in $\mathcal{H}$ with the maximum inner product value with $\mathbf{w}$:

$$\underset{j=1,\ldots,n}{\operatorname{argmax}} \mathbf{w}^T \mathbf{h}_j.$$

For instance, in matrix completion model for recommender systems, $\mathcal{H}$ is the latent features for items, $\mathbf{w}$ is the latent feature for a particular user, and the inner product $\mathbf{w}^T\mathbf{h}_j$ reflects the user's interest in the j-th item. Therefore recommending top items is equivalent to solving the MIPS problem.

A naive linear search procedure that computes the inner product between $\mathbf{w}$ and all $\{\mathbf{h}_j\}_{j=1}^n$ would require $O(nd)$ time, which is too slow for problems with large n. For example, in recommender systems, n (number of items) could easily go beyond a million and the number of features can also be very large. To solve this problem, many approximate algorithms have been proposed in the past to speed up MIPS, including LSH-based approaches and tree-based approaches (Bachrach et al., 2014; Neyshabur and Srebro, 2014; Ram and Gray, 2012; Shrivastava and Li, 2014a,b).

We study the sampling-based algorithm for speeding up the computation of MIPS. The main idea is to estimate the values of $\mathbf{w}^T\mathbf{h}_j$ using a fast sampling approach, and then rank the estimated scores to get the final candidates. Although sampling approach sounds reasonable for this task, it has not been successfully applied to MIPS due to the following reasons: 1) the hardness of handling negative values, 2) each sample has to be drawn efficiently without explicitly computing $\mathbf{w}^T\mathbf{h}_j$, and 3) lack of theoretical guarantee. By Cohen and Lewis (1999), a sampling-based approach was first discussed, but their algorithm only works for non-negative data matrices and so could not be used in most of the real applications. Recently Ballard et al. (2015) proposed a similar sampling approach for Maximum All-pairs Dot-Product (MAD) search problem, but also due to the hardness of handling negative values, they are only able to rank inner products based on absolute value of inner product $|\mathbf{w}^T\mathbf{h}_j|$. Moreover, their analysis works only for positive matrices.

In this paper, we propose a novel Sampling-MIPS algo-

Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89.

rithm which can be used in not only non-negative cases, but also general cases with both positive and negative values. Our algorithm only requires $O(1)$ time for generating each sample, and we prove that the algorithm will output the correct ranking when sample size is large enough. Furthermore, our algorithm can be used for sparse data with still $O(1)$ time complexity per sample and space complexity proportional to number of nonzero elements in $\mathcal{H}$. Experimental results show that our proposed algorithm is superior compared to other state-of-the-art methods.

Notations: Throughout the paper, $\mathbf{w} \in \mathbb{R}^d$ is the query vector, $\mathcal{H} = \{\mathbf{h}_1, \dots, \mathbf{h}_n\}$ are the n vectors in the database. For simplicity, we use a n-by-d matrix $H = [\mathbf{h}_1 \dots \mathbf{h}_n]^T$ to denote the database, and $h_{jt} = (\mathbf{h}_j)_t$ is the t-th feature of j-th vector.

2 Related Work

2.1 Previous MIPS Algorithms

It has been shown by Bachrach et al. (2014) that the MIPS problem can be reduced to Nearest Neighbor Search (NNS) by adding one more dimension to the features. Based on this reduction, NNS algorithms such as Locality Sensitive Hashing (LSH) (Indyk and Motwani, 1998) and PCA-tree (a modification of KD-tree) (Bachrach et al., 2014; Ram and Gray, 2012) can be used for solving MIPS. Neyshabur and Srebro (2014); Shrivastava and Li (2014a,b) proposed similar ways to reduce MIPS to NNS and solve the resulting problem using LSH. In the experiments, we show our proposed algorithm is better than both LSH and tree based approaches. Furthermore, as pointed out by Yu et al. (2017), these algorithms have to pre-specify number of hashings and tree splits, thus cannot control the trade-off between speedup and precision in run time. In comparison, our algorithm can easily control this trade-off by changing number of samples used in run time.

Concurrent to this work there are some other recent methods proposed for MIPS. Zhang et al. (2018) also applied the same MIPS-to-NNS reduction and then solve the resulting NNS problem by a graph-based approach (Malkov and Yashunin, 2018). Chen et al. (2019) proposed another gumbel clustering approach for MIPS in NLP applications.

2.2 Greedy-MIPS

Most recently, Yu et al. (2017) developed a deterministic algorithm called Greedy-MIPS. Their algorithm is mainly based on the following assumption:

$$\mathbf{w}^T\mathbf{h}_j > \mathbf{w}^T\mathbf{h}_i \;\Leftrightarrow\; \max_{t=1,\dots,d} w_t h_{jt} > \max_{t=1,\dots,d} w_t h_{it}, \quad (1)$$

which means when $\mathbf{w}^T\mathbf{h}_j = \sum_{t=1}^d w_t h_{jt}$ is large, at least one element in the summation will also be large. Based on this assumption they proposed a fast way to construct candidate set based on the ranking of $\max_{t=1,\dots,d} w_t h_{jt}$ for each j, and then use a linear search to get the final ranking among candidate set.

Clearly, Assumption (1) does not hold for general cases and Greedy-MIPS does not have performance guarantee in general. Nevertheless, Yu et al. (2017) showed that this Greedy-MIPS algorithm is quite powerful on real datasets. In Section 5.4, we conduct a comparison with Greedy-MIPS under different scenarios, showing that although Greedy-MIPS performs well on real datasets, when assumption (1) is violated it will produce poor results. This suggests that using Greedy-MIPS is unsafe. In contrary, our algorithm is guaranteed to output the correct top-K elements when the sample size is large enough.

2.3 Sampling-based approaches for Maximum Squared Inner Product Search

Our algorithm is the first sampling-based algorithm designed for MIPS with general input matrices. All the previous approaches are either restricted to non-negative input matrices or designed for another problem called Maximum Squared Inner Product Search (MSIPS).

Idea of sampling method for inner product search was firstly proposed by (Cohen and Lewis, 1999). The intuition behind this idea is to sample index j with probability $P(j|\mathbf{w}) \propto \mathbf{w}^T\mathbf{h}_j$. Unfortunately, this approach requires all the elements of $\mathbf{w}$ and H to be non-negative in order to define such a probability distribution, thus the applicability of (Cohen and Lewis, 1999) is very limited. Recently, a similar approach called diamond sampling was extended by Ballard et al. (2015). However, also due to the need of defining a valid probability distribution, this method can only identify indices with the largest $|\mathbf{w}^T\mathbf{h}_j|$ or $(\mathbf{w}^T\mathbf{h}_j)^2$, which is called MSIPS problem. If both $\mathbf{w}$ and H have non-negative entries or more generally, $\mathbf{w}^T\mathbf{h}_j \geq 0$, $\forall j$, MSIPS can be reduced to MIPS problem. But in reality, we frequently have matrices and queries with both positive and negative values. Thus to develop a proper sampling approach for MIPS with negative values with well-founded theory remains a challenge.

3 Proposed Algorithm

Similar to many previous methods, our algorithm has two phases. Before observing any query, we pre-process the database H and construct some data structure to facilitate the later sampling procedure. This is called the pre-preocessing phase and the time will not be counted into the per-query cost. In the query-dependent phase, the MIPS algorithm is conducted using both query $\mathbf{w}$ and the pre-constructed data structure, and our main goal is to reduce the per-query cost in this phase.

Without loss of generality, in the paper, we assume that none of $\{\mathbf{h}_j\}$ and $\mathbf{w}$ is the zero vector. In the following we first present Sampling-MIPS algorithm for non-negative values and then show how to handle problems with both positive and negative values.

3.1 Sampling-MIPS for non-negative values

First we assume the database H and query $\mathbf{w}$ only have non-negative values. The main idea is to draw samples from the marginal distribution, where each index j is sampled with probability

$$P(j|\mathbf{w}) \propto \mathbf{w}^T \mathbf{h}_j.$$

If we can draw samples according to the above distribution, then with high probability the candidate with the maximum inner product value will be drawn most frequently. Moreover, we have total freedom to control the search quality in query time, since the quality can keep being improved by drawing more samples.

However, computing the distribution of $P(j|\mathbf{w})$ for all $j = 1, \ldots, n$ will be extremely time-consuming. In order to sample from this marginal distribution without explicitly forming them, we define a joint distribution for the pair (j, t) over the region $\{1, 2, \cdots, n\} \times \{1, 2, \cdots, d\}$. Let $P(j, t|\mathbf{w}) \propto w_t h_{jt}$, then the marginal distribution of j can be expressed as

$$P(j|\mathbf{w}) = \sum_{t=1}^{d} P(j, t|\mathbf{w}) = \sum_{t=1}^{d} P(t|\mathbf{w}) P(j|t, \mathbf{w}), \quad (2)$$

where

$$P(t|\mathbf{w}) = \sum_{j=1}^{n} P(j, t|\mathbf{w}) \propto \sum_{j=1}^{n} w_t h_{jt} = w_t s_t, \quad (3)$$

$$P(j|t, \mathbf{w}) = \frac{P(j, t|\mathbf{w})}{P(t|\mathbf{w})} \propto \frac{w_t h_{jt}}{\sum_{j=1}^{n} w_t h_{jt}} = \frac{h_{jt}}{\sum_{j=1}^{n} h_{jt}} = \frac{h_{jt}}{s_t}. \quad (4)$$

Note that we define $s_t = \sum_{j=1}^{n} h_{jt}$ to simplify the notation and we assume $s_t \neq 0, \forall t$. From equation (4), we can see that $P(j|t, \mathbf{w})$ is irrelevant to $\mathbf{w}$, so we can rewrite $P(j|t, \mathbf{w})$ as $P(j|t)$.

Based on (3) and (4), to draw a sample from $P(j|\mathbf{w})$, we can first draw t from a multinomial distribution, $P(t|\mathbf{w}) = \text{multinomial}([\frac{w_1 s_1}{\sum_{t=1}^{d} w_t s_t}, \cdots, \frac{w_d s_d}{\sum_{t=1}^{d} w_t s_t}])$, and then draw a sample for j from multinomial distribution $P(j|t) = \text{multinomial}([\frac{h_{1t}}{s_t}, \cdots, \frac{h_{nt}}{s_t}])$. The procedure is illustrated in Figure 1. In Figure 1, sampling $t \sim P(t|\mathbf{w})$ gives us $t = 4$, so we focus on the 4-th column of matrix H and sample $j \sim P(j|t = 4)$ which gives us $j = 2$, and this sampling procedure is equivalent to sampling j from the marginal distribution $P(j|\mathbf{w})$.

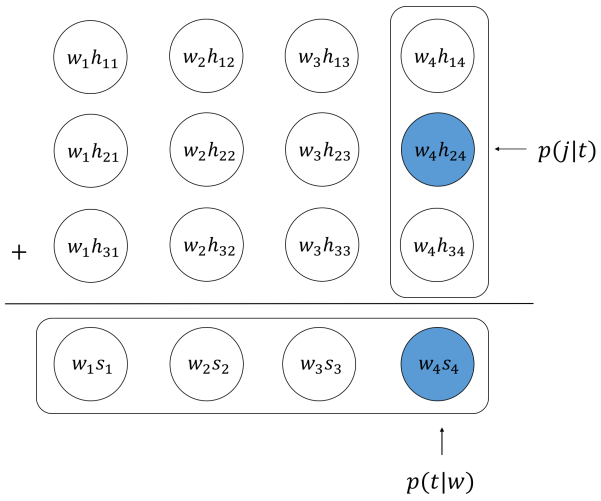

Figure 1: Illustration of the sampling procedure.

In order to quickly draw samples, we use alias method proposed by Walker (1977). For a given k-dimensional multinomial distribution, constructing an alias table requires $O(k)$ time and $O(k)$ space, and the alias table will allow us to draw each sample with only $O(1)$ time. From the above definitions, we can see that the distribution for sampling index j is irrelevant to $\mathbf{w}$, which means we can construct the alias table for $P(j|t)$ in the pre-processing phase. We present our algorithm for pre-processing phase in Algorithm 1.

Algorithm 1 Query-independent pre-processing

1: $s_t \leftarrow \sum_{j=1}^{n} h_{jt}, \forall t \in \{1, \cdots, d\}$
2: Use alias table to construct
$P(j|t) \leftarrow \text{multinomial}([h_{1t}, \cdots, h_{nt}]), \forall t$

In the query-dependent phase, we should first construct the alias table for $P(t|\mathbf{w})$, which requires $O(d)$ time. This will allow us to draw samples from $P(t|\mathbf{w})$ in constant time later. We show the procedure of query-dependent candidate screening in Algorithm 2. We use a vector of length n to count how many times each index is sampled. From line 3 to line 7 in Algorithm 2, we draw B samples and for each one we sample j using the two-step procedure mentioned above. Each time an index j is sampled, we increase the count of index j by 1.

After the sampling process, we sort all the n candidates according to the ranking of x_j's and take top-C of them. We perform a naive-MIPS on these selected C candidates, i.e., we compute these C inner products and find which candidates have the maximum or top-K values. Details are shown in Algorithm 3.

Since this method is built on the fact that we sample index j with probability proportional to $\mathbf{w}^T \mathbf{h}_j$, we know that the algorithm will succeed (output the real top-K elements)

Algorithm 2 Query-dependent candidate screening

Use alias table to construct $P(t|\mathbf{w}) \leftarrow \text{multinomial}([w_1 s_1, \cdots, w_d s_d])$ $\cdots O(d)$
$\mathbf{x} = [x_1, x_2, \cdots, x_n] \leftarrow [0, \ldots, 0]$
for $b = 1, \cdots, B$ **do** $\cdots O(B)$
    Use alias method to sample $t \sim P(t|\mathbf{w})$
    Use alias method to sample $j \sim P(j|t)$
    $x_j \leftarrow x_j + 1$
end for
Pass $\mathbf{x}$ to Algorithm 3

Algorithm 3 Final Prediction Phase

Find the top-C elements in $\mathbf{x}$, i.e., $|\mathcal{C}(\mathbf{w})| = C$ and $\mathcal{C}(\mathbf{w}) \leftarrow \{j | x_j \geq x_{j'}, \forall j' \notin \mathcal{C}(\mathbf{w})\}$ $\cdots O(B)$
for $j \in \mathcal{C}(\mathbf{w})$ **do** $\cdots O(Cd)$
    Compute inner product $\mathbf{w}^T \mathbf{h}_j$
end for
Output: indexes of the top-K elements of $\{\mathbf{w}^T \mathbf{h}_{j_1}, \cdots, \mathbf{w}^T \mathbf{h}_{j_C}\}$ $\cdots O(C)$

with high probability if the sample size B is sufficiently large. We will provide the detailed theoretical analysis in Section 4.

3.2 Handling both positive and negative values

Next we discuss how to handle the case when both H and $\mathbf{w}$ can be negative. To extend our approach to the general case, we need to make sure that we don't violate the principle of getting larger x_j for index j with larger inner product. In order to achieve this goal, we keep the probability distribution to be non-negative by taking absolute values of all the elements of H and $\mathbf{w}$, that is, $P(j, t|\mathbf{w}) \propto |w_t h_{jt}|$. We then use the following sampling procedure to estimate the expectation of $E_{P(j,t|\mathbf{w})}[\text{sgn}(w_t h_{jt})]$, which is proportional to $\mathbf{w}^T \mathbf{h}_j$ (see Lemma 1 in Section 4). In the pre-processing phase, we build alias tables for the probability distribution $P(j|t) = \frac{|h_{jt}|}{\sum_{j=1}^n |h_{jt}|}$ for each $t \in \{1, \ldots, d\}$. Then in the query-dependent phase, we construct alias table with $P(t|\mathbf{w}) \propto \sum_{j=1}^n |w_t h_{jt}|$ to sample t given the current query. The marginal distribution then becomes

$$\begin{aligned} P(j|\mathbf{w}) &= \sum_{t=1}^d P(j, t|\mathbf{w}) \\ &= \sum_{t=1}^d P(j|t) P(t|\mathbf{w}) \\ &\propto \sum_{t=1}^d |w_t h_{jt}|. \end{aligned} \tag{5}$$

We sample t and then j for B times, but each time a pair of index (j, t) is sampled, we increase x_j by $\text{sgn}(w_t h_{jt})$ instead of 1 for non-negative cases. The detailed procedure and time complexity of each step are shown in Algorithm 4. The total time complexity is then $O(d + B + Cd)$, while the naive time complexity is $O(nd)$. Usually, the budget size C is much smaller than n and in practice $B \approx n$ gives good results, so the proposed algorithm is much faster than the naive search approach.

Algorithm 4 Sampling-MIPS for general cases

Pre-processing:
$s_t \leftarrow \sum_{j=1}^n |h_{jt}|, \forall t = 1, \cdots, d$
Use alias table to construct $P(j|t) \leftarrow \text{multinomial}([|h_{1t}|, \cdots, |h_{nt}|]), \forall t$
Candidate Screening:
Use alias table to construct $P(t|\mathbf{w}) \leftarrow \text{multinomial}([|w_1 s_1|, \cdots, |w_d s_d|])$ $\cdots O(d)$
$\mathbf{x} = [x_1, x_2, \cdots, x_n] \leftarrow [0, \ldots, 0]$
Specify sample size B
for $b = 1, \cdots, B$ **do** $\cdots O(B)$
    Use alias method to sample $t \sim P(t|\mathbf{w})$
    Use alias method to sample $j \sim P(j|t)$
    $x_j \leftarrow x_j + \text{sgn}(w_t h_{jt})$
end for
Final Prediction Phase: $\cdots O(B + Cd)$
See Algorithm 3.

The key idea of Sampling-MIPS is to construct the "scores" (x_j) to estimate the n inner products. Index with larger inner product will have larger "score" after the entire sampling procedure. Yet directly sampling j is hard, we define a probability distribution for (j, t) and sample indexes using the two-step procedure. This method is expected to succeed with high probability if sampled enough times. When B is large enough, the result won't change a lot. Intuitively, we know that the required best sample size B to achieve certain accuracy relies on the relative gap between the n inner products. We will give a detailed analysis of the error bound for Sampling-MIPS in Section 4.

4 Mathematical Analysis of Sampling-MIPS

In the following we analyze the Sampling-MIPS algorithm for general case (with both positive and negative values). Due to the space limits, the proofs are deferred to the supplementary material. Recall that for the general case, the marginal distribution of j is $P(j|\mathbf{w}) \propto \sum_{t=1}^d |w_t h_{jt}|$, and our goal is to prove that $E[x_j]$ will be proportional to $\mathbf{w}^T \mathbf{h}_j$. For convenience, we define $x_{jb} = \text{sgn}(w_t h_{jt})$ if index pair (j, t) is sampled in the b-th sampling step, and $x_{jb} = 0$ otherwise. Based on this definition, we have $x_j = \sum_{b=1}^B x_{jb}$. The mean and variance of x_{jb} and x_j can be established by the following lemmas.

Lemma 1. *Define* $S = \sum_{t=1}^d \sum_{j=1}^n |w_t h_{jt}|$ *and* $c_j = \mathbf{w}^T \mathbf{h}_j$, *then* $E[x_{jb}] = \frac{c_j}{S}$ *and* $E[x_j] = \frac{B c_j}{S}$.

Lemma 2. *Define* $a_j = \sum_{t=1}^{d} |w_t h_{jt}|$, *then* $E[(x_{jb} - x_{mb})^2] = \frac{a_j + a_m}{S}, \forall j \neq m$. *Therefore, from Lemma 1, we have*

$$Var(x_{jb} - x_{mb}) = \frac{a_j + a_m}{S} - \frac{(c_j - c_m)^2}{S^2}$$

and $Var(x_j - x_m) = BVar(x_{jb} - x_{mb})$.

Theorem 1. *If index* m *has the maximum inner product, we define for* $j \in \{j : c_j < c_m\}$

$$\lambda_j = \frac{c_m - c_j}{S},$$
$$T_j = \frac{(S + c_m - c_j)(c_m - c_j)}{S^2},$$
$$M_j = \frac{S^2}{(S + c_m - c_j)^2},$$
$$\sigma_j^2 = Var(x_{jb} - x_{mb}).$$

Also let $\lambda = \min_{j \in \{c_j < c_m\}} \lambda_j$, $n' = \#\{j : c_j < c_m\}$ *and assume* $n' \neq 0$. *If sample size* B *satisfies the following equation for all* $j \in \{j : c_j < c_m\}$,

$$B \geq \frac{\log \frac{n'}{\delta}}{M_j \sigma_j^2 \left[(1 + \frac{T_j}{\sigma_j^2}) \log(1 + \frac{T_j}{\sigma_j^2}) - \frac{T_j}{\sigma_j^2} \right]}, \quad (6)$$

implying that when $B \sim O(\frac{d}{\lambda} \log n')$, *then with error probability* $\delta \in (0, 1)$ *we have*

$$P(x_m > x_j, \forall j \in \{c_j < c_m\}) \geq 1 - \delta.$$

This Theorem states that when number of samples $B = O(\frac{d}{\lambda} \log n')$, Sampling-MIPS will be able to distinguish candidate vector m from the rest candidates with smaller inner products. When candidate vector m has the maximum inner product, n' can be replaced by n, so Theorem 1 suggests that Sampling-MIPS can get precise results when $B = O(\frac{d}{\lambda} \log n)$. Also, when the probability for sampling different indexes are very close, we need larger sample size to distinguish them. This is reflected in Theorem 1 via the numerator of λ, $\min_{\{j:c_j<c_m\}}(c_m - c_j)$. Note that $\lambda < 1$ here.

Theorem 2. *Assume* $w_t h_{jt} \geq 0, \forall (j, t) \in \{1, \ldots, n\} \times \{1, \ldots, d\}$ *and* $c_j \overset{iid}{\sim} Exp(\beta)$. *In Sampling-MIPS algorithm, if we take* $C = B$, *where* C *means we calculate inner products of indexes with top-*C *scores, then when* $B \geq \frac{\rho}{\alpha} n$, *where* $\alpha \in (0, \frac{1}{2})$ *and* $\rho \in (0, \frac{\alpha d(n-1)}{(d+1)n})$,

$$P(\textit{not identifying maximum inner product}) \leq O\left(\frac{1}{n^\rho}\right).$$

This result can also be generalized to inner products *i.i.d.* sampled from heavy-tailed distributions.

Definition 1. *The distribution of a random variable X with c.d.f.* $F(x)$ *is said to have a heavy (right) tail if*

$$\lim_{x \to +\infty} e^{\mu x}(1 - F(x)) = +\infty, \ \forall \mu > 0. \quad (7)$$

Note: Heavy-tailed distribution includes many common distributions like log-normal, log-cauchy, Lévy and Pareto distribution, etc.

Corollary 1. *Assume* $w_t h_{jt} \geq 0, \forall (j, t) \in \{1, \ldots, n\} \times \{1, \ldots, d\}$. *If* $\mathcal{F}$ *is a continuous, non-negative, heavy-tailed distribution,* $c_j \overset{iid}{\sim} \mathcal{F}$ *and* $E[c_j^2] < \infty$, *then when* $B \geq \frac{\rho}{\alpha} n$, *where* $\alpha \in (0, 1)$ *and* $\rho \in (0, \frac{\alpha d(n-1)}{(d+1)n})$, *take* $C = B$ *in Sampling-MIPS algorithm, we have*

$$P(\textit{not identifying maximum inner product}) \leq O\left(\frac{1}{n^\rho}\right).$$

Theorem 2 and Corollary 1 imply that we only need sample size B to be $\frac{\rho}{\alpha} n$ to make sure that the maximum inner product to be identified. The total time complexity of Sampling-MIPS is $O(d + B + Bd)$ and the time complexity of Naive-MIPS is $O(nd)$. When $B = \frac{\rho}{\alpha} n$,

$$d + B + Bd = d + \frac{\rho}{\alpha}(d+1)n < d + \frac{d(n-1)}{(d+1)n} n(d+1) = nd.$$

Therefore, the conditions of parameter α and ρ ensures that Sampling-MIPS is more efficient than Naive-MIPS.

Note that in Theorem 2, α can be any number in $(0, \frac{1}{2})$, but Corollary 1 only requires $\alpha \in (0, 1)$. ρ can be any number in $(0, \frac{\alpha d(n-1)}{(d+1)n})$. Sampling-MIPS algorithm is faster with smaller ρ, but can also yield larger error probability. So ρ should be chosen according to the trade-off between efficiency and tolerance of error.

5 Experimental Results

In this section, we compare the performance of Sampling-MIPS with other state-of-the-art algorithms. In order to have a fair comparison, all the algorithms are implemented in highly optimized C++ code using the platform provided by Yu et al. (2017). We also follow the same evaluation criteria used in (Yu et al., 2017) to measure the precision and speedup of MIPS algorithms. To compute the precision, we randomly select 2,000 queries and compute the average of $prec@K$ for $K = 1, 5, 10$, where $prec@K$ is defined as:

$$prec@K = \frac{\#\{\text{selected top-}K\} \cap \{\text{true top-20}\}}{K}.$$

The true top-20 candidates are identified by naive MIPS approach, and the selected top-K candidates are identified by approximate MIPS algorithms. For the speed of MIPS algorithms, we report the speedup of each algorithm over the naive linear search approach (computing all the inner

products). All the experiments are conducted on a server with Intel Xeon E5-2640 CPU and only one core is used for the experiments.

5.1 Datasets and parameter settings

We compare the results on several real recommender system datasets, including MovieLens-20M (Harper and Konstan, 2016) and Netflix. MovieLens-20M contains $27,278$ movies and $138,493$ users while Netflix contains $17,770$ items and $480,189$ users. For each dataset, we obtain the low-rank matrices W and H by matrix factorization model using the open source matrix factorization package LIBPMF (Yu et al., 2014a). We use $\lambda = 0.05$ for both MovieLens-20M and Netflix datasets to get latent features W and H with rank $d = 50, 100, 200$.

To test the performance of algorithms under different scenarios, we also conduct experiments on synthetic datasets. The candidate matrix H and query vector $\mathbf{w}$ are generated from $\mathcal{N}(0, 10)$, with $n = 200000$ and $d \in \{50, 100, 200\}$ (used in Figure 3).

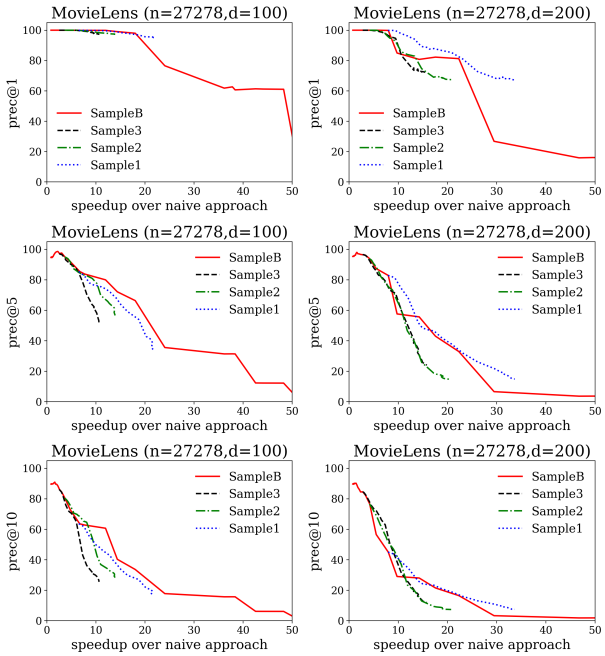

Figure 2: Comparison of four different parameter settings of Sampling-MIPS on MovieLens data.

5.2 Sampling-MIPS with different parameters

In Sampling-MIPS, there are two parameters that determine the trade-off between speedup and precision: sample size B and the size of budget $\mathcal{C}(w)$ denoted as C. We describe four methods of choosing B and C in this paper:

- SampleB: Set $C = B$ and B ranges from $0.95n$ to $0.001n$.
- Sample1: Fix sample size $B = 0.1n$ and set $C = r \times B$, where r ranges from 1 to 0.06.
- Sample2: Fix sample size $B = 0.2n$ and set $C = r \times B$, where r ranges from 1 to 0.06.
- Sample3: Fix sample size $B = 0.3n$ and set $C = r \times B$, where r ranges from 1 to 0.06.

The comparison of SampleB, Sample1, Sample2 and Sample3 on MovieLens-20M data are shown in Figure 2. In general we found that they are similar in most cases, but the approach of SampleB is the most stable one, so we choose SampleB and compare it with other MIPS algorithms in the following subsections.

5.3 Experimental results on real datasets

We report the comparisons on MovieLens-20M, Netflix and synthetic data with different dimensionality in Figure 3. It is easy to see that Sampling-MIPS outperforms PCA-MIPS, LSH-MIPS and Diamond-MSIPS in most of the cases. It is worthwhile to mention that on MovieLens-20M data with $d = 50, 100$, when speedup is 10, Sampling-MIPS yields $prec@1 \approx 100\%$ and $prec@5 \approx 80\%$, while none of the other methods can reach $prec@1 \geq 50\%$ and $prec@5 \geq 20\%$. See tables in the supplementary material for more details.

5.4 Comparison to Greedy-MIPS

Greedy-MIPS is the most competitive algorithm among all the previous approaches in practice. Although it does not have a theoretical guarantee, it usually yields better results than Sampling-MIPS and all the other approaches on real datasets (see second row of Figure 4). This means Assumption (1), although clearly not true in general, is valid to some extent on real datasets. However, as we will show in the following experiments, when Assumption (1) does not hold, Greedy-MIPS will perform poorly.

In the first row of Figure 4, we construct the synthetic data with irregular pattern which violates Assumption (1). The i-th row of candidate matrix H is generated from $\mathcal{N}(\frac{200000}{i}, \frac{i}{10})$ and query vector $\mathbf{w}$ is generated from $\mathcal{N}(1, 0.1)$, with $n = 200000$ and $d = 2000$. The results are shown in the first row of Figure 4. Clearly, in this setting Assumption (1) does not hold, so $prec@10$ of Greedy-MIPS drops to 40% even when speedup ≤ 5. In comparison, our algorithm still gets almost perfect precision under the same speedup. In comparison, if we generate synthetic datasets with all the rows in H follow the same uniform distribution, the results in the third row of Figure 4 shows Greedy-MIPS performs well in this case since Assumption (1) is mostly satisfied.

5.5 Implementation for sparse matrices

Sampling-MIPS draws samples based on a probability distribution of each entry. As zero entries will never be sampled, we can improve the efficiency of sampling and reduce the memory cost by only considering the non-zeros. Therefore, it is nature that Sampling-MIPS will work especially good when dealing with sparse matrices. We utilize Eigen (version 3) (Guennebaud et al., 2010) to input data and implement Sampling-MIPS algorithm in a smarter way by only dealing with non-zero entries.

To test our implementation, we generate sparse candidate matrices $H \sim Uniform(0, 1)$ and sparse query vectors $\mathbf{w} \sim \mathcal{N}(0, 10)$ with $n = 20000$ and $d \in \{200, 1000\}$. The experimental results in Figure 5 demonstrate that our algorithm performs well on sparse data, while many existing methods such as Greedy-MIPS cannot handle sparse data.

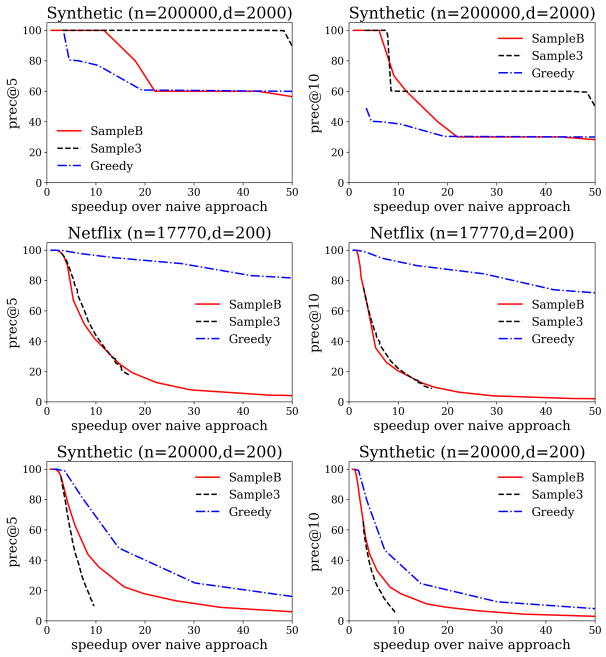

Figure 4: Comparison of Sampling-MIPS and Greedy-MIPS on Netflix and synthetic datasets.

6 Conclusion

We propose a Sampling-MIPS algorithm for approximate maximum inner product search. The core idea is to develop a novel way to sample from the distribution according to the magnitude of inner product, but without explicitly constructing such distribution. Each sample only costs constant time, and we show that the algorithm can recover the true ranking of inner products when sample size is large enough. Experimental results on real and synthetic datasets show that our algorithm significantly outperforms most of previous approaches including PCA-MIPS, LSH-MIPS and Diamond-MSIPS. Compared to Greedy-MIPS, our algorithm does not rely on any vulnerable assumptions and is safer to be applied to all kinds of datasets. Moreover, our algorithm can be perfectly adapted to data of sparse format.

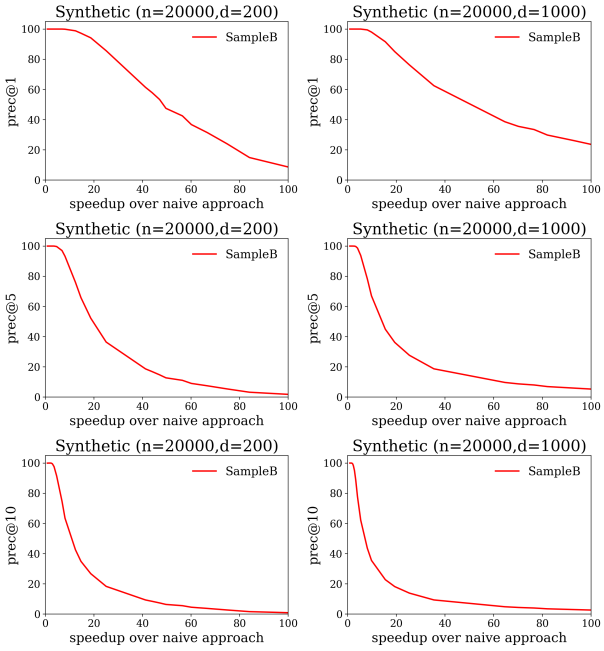

Figure 5: Results of Sampling-MIPS for data of sparse format with $n = 20000$ and $d \in \{200, 1000\}$.

7 Acknowledgement

We are grateful to Xiaodong Li and James Sharpnack for helpful comments. This work was partially supported by NSF IIS-1719097, Intel, Google Cloud and AITRICS.

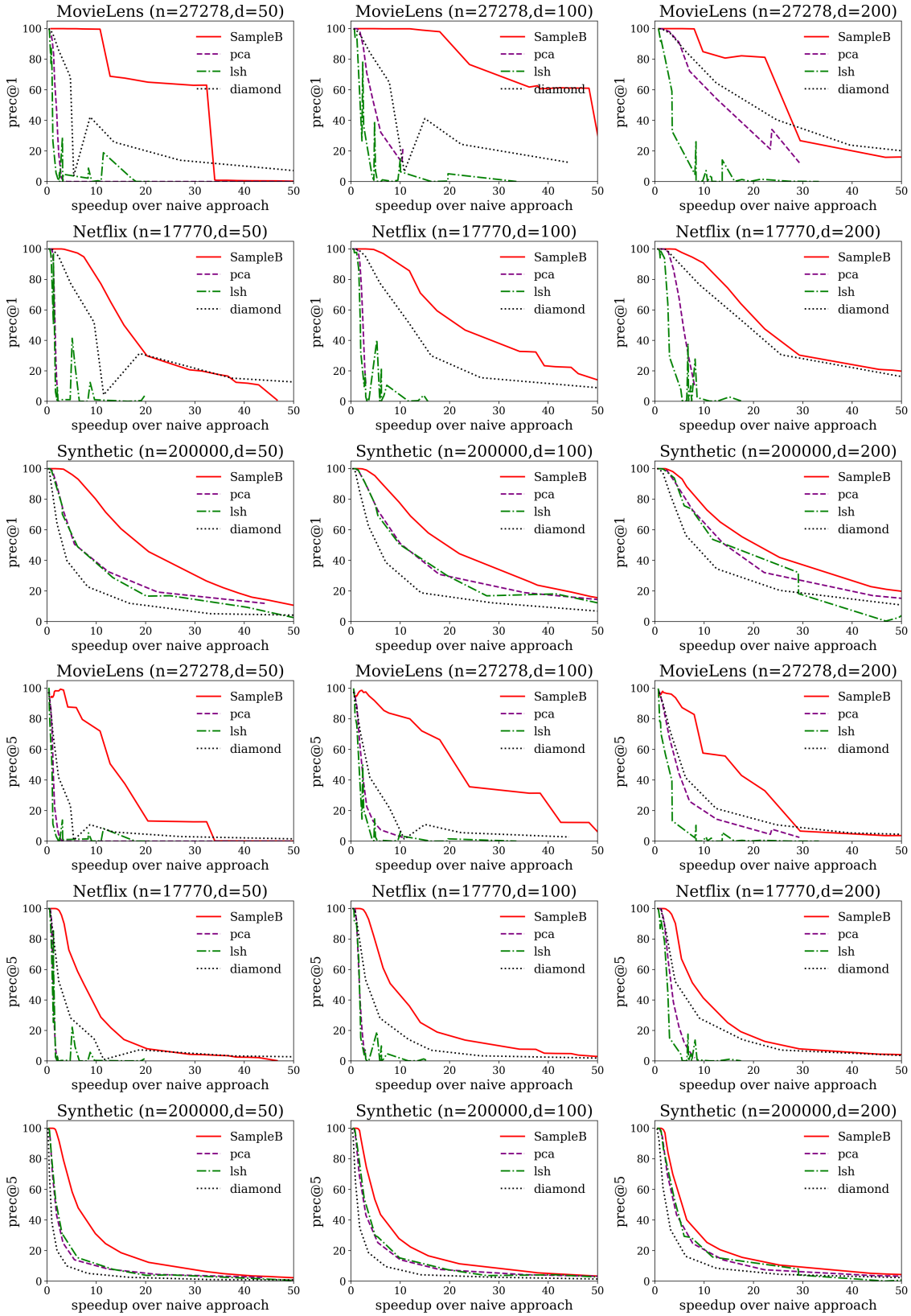

Figure 3: Comparison of Sampling-MIPS to PCA-MIPS, LSH-MIPS, Diamond-MSIPS on MovieLens, Netflix and synthetic datasets. The synthetic datasets have their H and $\mathbf{w}$ generated from $\mathcal{N}(0, 10)$.

References

Rohit Babbar and Bernhard Schölkopf. Dismec: distributed sparse machines for extreme multi-label classification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 721–729. ACM, 2017.

Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 257–264. ACM, 2014.

Grey Ballard, Tamara G Kolda, Ali Pinar, and C Seshadhri. Diamond sampling for approximate maximum all-pairs dot-product (mad) search. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 11–20. IEEE, 2015.

George Bennett. Probability inequalities for the sum of independent random variables. *Journal of the American Statistical Association*, 57(297):33–45, 1962.

Patrick H Chen, Si Si, Sanjiv Kumar, Yang Li, and Cho-Jui Hsieh. Learning to screen for fast softmax inference on large vocabulary neural networks. In *ICLR*, 2019.

Edith Cohen and David D Lewis. Approximating matrix multiplication for pattern recognition tasks. *Journal of Algorithms*, 30(2):211–252, 1999.

Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19, 2016.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613, 1998.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

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

Behnam Neyshabur and Nathan Srebro. On symmetric and asymmetric lshs for inner product search. *arXiv preprint arXiv:1410.5518*, 2014.

Parikshit Ram and Alexander G Gray. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 931–939. ACM, 2012.

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

Anshumali Shrivastava and Ping Li. Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips). *arXiv preprint arXiv:1410.5410*, 2014b.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Alastair J Walker. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software (TOMS)*, 3(3): 253–256, 1977.

Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. Parallel matrix factorization for recommender systems. *Knowledge and Information Systems*, 41(3):793–819, 2014a.

Hsiang-Fu Yu, Prateek Jain, Purushottam Kar, and Inderjit Dhillon. Large-scale multi-label learning with missing labels. In *International conference on machine learning*, pages 593–601, 2014b.

Hsiang-Fu Yu, Cho-Jui Hsieh, Qi Lei, and Inderjit S Dhillon. A greedy approach for budgeted maximum inner product search. In *Advances in Neural Information Processing Systems*, pages 5459–5468, 2017.

Minjia Zhang, Wenhan Wang, Xiaodong Liu, Jianfeng Gao, and Yuxiong He. Navigating with graph representations for fast and scalable decoding of neural language models. In *Advances in Neural Information Processing Systems*, pages 6311–6322, 2018.

8 Supplementary Materials

8.1 Proofs

8.1.1 Proof of Lemma 1

Proof. We use *law of iterated expectation* to prove the conclusion. We first compute

$$\begin{aligned} E[x_{jb}|t] &= \text{sgn}(w_t h_{jt}) P(j|t) \\ &= \text{sgn}(w_t h_{jt}) \frac{|h_{jt}|}{\sum_{j=1}^n |h_{jt}|} \\ &= \text{sgn}(w_t h_{jt}) \frac{|w_t h_{jt}|}{\sum_{j=1}^n |w_t h_{jt}|} \\ &= \frac{w_t h_{jt}}{\sum_{j=1}^n |w_t h_{jt}|}. \end{aligned}$$

This gives us:

$$\begin{aligned} E[x_{jb}] &= E[E[x_{jb}|t]] = E[\frac{w_t h_{jt}}{\sum_{j=1}^n |w_t h_{jt}|}] \\ &= \sum_{t=1}^d \frac{w_t h_{jt}}{\sum_{j=1}^n |w_t h_{jt}|} P(t|\mathbf{w}) \\ &= \sum_{t=1}^d \frac{w_t h_{jt}}{\sum_{j=1}^n |w_t h_{jt}|} \frac{\sum_{j=1}^n |w_t h_{jt}|}{\sum_{t=1}^d \sum_{j=1}^n |w_t h_{jt}|} \\ &= \frac{\sum_{t=1}^d w_t h_{jt}}{\sum_{t=1}^d \sum_{j=1}^n |w_t h_{jt}|} \\ &= \frac{\mathbf{w}^T \mathbf{h}_j}{\sum_{t=1}^d \sum_{j=1}^n |w_t h_{jt}|} = \frac{c_j}{S}. \end{aligned}$$

Therefore, $E[x_j] = E[\sum_{b=1}^B x_{jb}] = \frac{Bc_j}{S}$. □

8.1.2 Proof of Lemma 2

Proof. We still use *law of iterated expectation* to prove the conclusion and it is basically very similar to the proof of Lemma 1.

$$\begin{aligned} & E[(x_{jb} - x_{mb})^2|t] \\ = \; & [\text{sgn}(w_t h_{jt}) - 0]^2 P(j|t) + [0 - \text{sgn}(w_t h_{mt})]^2 P(m|t) \\ = \; & [\text{sgn}(w_t h_{jt})]^2 \frac{|h_{jt}|}{\sum_{j=1}^n |h_{jt}|} \\ & + [\text{sgn}(w_t h_{mt})]^2 \frac{|h_{mt}|}{\sum_{j=1}^n |h_{jt}|} \\ = \; & [\text{sgn}(w_t h_{jt})]^2 \frac{|w_t h_{jt}|}{\sum_{j=1}^n |w_t h_{jt}|} \\ & + [\text{sgn}(w_t h_{mt})]^2 \frac{|w_t h_{mt}|}{\sum_{j=1}^n |w_t h_{jt}|} \\ = \; & \frac{|w_t h_{jt}| + |w_t h_{mt}|}{\sum_{j=1}^n |w_t h_{jt}|}. \end{aligned}$$

This gives us:

$$\begin{aligned} E[(x_{jb} - x_{mb})^2] &= E\{E[(x_{jb} - x_{mb})^2|t]\} \\ &= \sum_{t=1}^d \frac{|w_t h_{jt}| + |w_t h_{mt}|}{\sum_{j=1}^n |w_t h_{jt}|} P(t|\mathbf{w}) \\ &= \frac{\sum_{t=1}^d (|w_t h_{jt}| + |w_t h_{mt}|)}{\sum_{t=1}^d \sum_{j=1}^n |w_t h_{jt}|} \\ &= \frac{a_j + a_m}{S}. \end{aligned}$$

Note that for each iteration b, $x_{jb} - x_{mb}$ is independently and identically distributed, so we have

$$\begin{aligned} \text{Var}(x_j - x_m) &= B\text{Var}(x_{jb} - x_{mb}) \\ &= B\left\{E[(x_{jb} - x_{mb})^2] - [E(x_{jb} - x_{mb})]^2\right\} \\ &= B\left[\frac{a_j + a_m}{S} - \frac{(c_j - c_m)^2}{S^2}\right]. \end{aligned}$$

□

8.1.3 Proof of Theorem 1

Proof. For some j, we know that x_{jb} and x_{mb} cannot be non-zero simultaneously. Therefore, all $(x_{jb} - x_{mb})$'s independently take values in $[-1, 1]$. We use Bennett's Inequality in (Bennett, 1962) to bound the probability of $P(x_j \geq x_m)$ for some $j \in \{c_j < c_m\}$.

$$\begin{aligned} P(x_j \geq x_m) &= P(x_j - x_m \geq 0) \\ &= P\left\{\sum_{b=1}^B (x_{jb} - x_{mb}) \geq 0\right\} \\ &= P(Y \geq y), \end{aligned} \tag{8}$$

where $Y_b = x_{jb} - x_{mb} - \frac{c_j - c_m}{S}$, $Y = \sum_{b=1}^B Y_b$ and $y = \frac{B(c_m - c_j)}{S}$. It is obvious that $Y_b \leq 1 - \frac{c_j - c_m}{S}$ almost surely. We denote $q = 1 - \frac{c_j - c_m}{S}$ and we compute a few quantities needed in Bennett's Inequality below:

$$\begin{aligned} & \Sigma_j^2 := BEY_b^2 = \text{Var}(x_j - x_m) = B\sigma_j^2, \\ & qy = B\frac{(S + c_m - c_j)(c_m - c_j)}{S^2} = BT_j, \\ & \frac{qy}{\Sigma_j^2} = \frac{T_j}{\sigma_j^2}, \\ & \frac{T_j}{\sigma_j^2} = \frac{(S + c_m - c_j)(c_m - c_j)}{S(a_j + a_m) - (c_m - c_j)^2}. \end{aligned} \tag{9}$$

From Bennett's Inequality, we can bound Equation (8):

$$\begin{aligned} & P(Y \geq y) \\ \leq \; & exp\left\{-\frac{\Sigma_j^2}{q^2}\left[(1 + \frac{qy}{\Sigma_j^2})\log(1 + \frac{qy}{\Sigma_j^2}) - \frac{qy}{\Sigma_j^2}\right]\right\} \\ = \; & exp\left\{-BM_j\sigma_j^2\left[(1 + \frac{T_j}{\sigma_j^2})\log(1 + \frac{T_j}{\sigma_j^2}) - \frac{T_j}{\sigma_j^2}\right]\right\} \end{aligned}$$

If we want $P(Y \geq y) \leq \frac{\delta}{n'}$, then it is equivalent to

$$BM_j\sigma_j^2\left[(1+\frac{T_j}{\sigma_j^2})\log(1+\frac{T_j}{\sigma_j^2})-\frac{T_j}{\sigma_j^2}\right] \geq \log\frac{n'}{\delta}.$$

Before proceeding to the result, we need to show that $(1+\frac{T_j}{\sigma_j^2})\log(1+\frac{T_j}{\sigma_j^2})-\frac{T_j}{\sigma_j^2} > 0$. Denote $v=\frac{T_j}{\sigma_j^2}$ and $g_1(v) = (1+v)\log(1+v)-v$. Then $g_1'(v) = 1+\log(1+v)-1 = \log(1+v) > 0$ when $v>0$. So $g_1(v) > g_1(0) = 0$ when $v>0$. It is not hard to see that $\frac{T_j}{\sigma_j^2} > 0$ from (9) for all $j \in \{c_j < c_m\}$. Therefore,

$$B \geq \frac{1}{M_j\sigma_j^2\left[(1+\frac{T_j}{\sigma_j^2})\log(1+\frac{T_j}{\sigma_j^2})-\frac{T_j}{\sigma_j^2}\right]}\log\frac{n'}{\delta}. \quad (10)$$

So far, we have proved that when B satisfies equation (6), $P(x_j \geq x_m) \leq \frac{\delta}{n'}$ for some $j \in \{j : c_j < c_m\}$. Since $\#\{j : c_j < c_m\} = n'$, we have when B satisfies equation (6) for all $j \in \{j : c_j < c_m\}$, the following holds:

$$\begin{aligned} & P(x_m > x_j, \forall j \in \{c_j < c_m\}) \\ = \;& 1 - P(x_m \leq x_j, \exists j \in \{c_j < c_m\}) \\ \geq \;& 1 - n'\frac{\delta}{n'} = 1-\delta. \end{aligned}$$

It is worthwhile to notice that $M_j \sim O(1)$, $T_j \sim O(\lambda)$ and $O(\lambda) \leq \sigma_j^2 \leq O(d\lambda)$ for all $j \in \{j : c_j < c_m\}$. Denote $g_2(\sigma_j^2, T_j) = (\sigma_j^2+T_j)\log(1+\frac{T_j}{\sigma_j^2}) - T_j$, then $\frac{\partial g_2}{\partial \sigma_j^2} = \log(1+\frac{T_j}{\sigma_j^2}) - \frac{T_j}{\sigma_j^2} < 0$. So $g_2(\sigma_j^2, T_j) \geq O\left(g_2(d\lambda, T_j)\right) = O(\frac{\lambda}{d})$. Therefore, the *r.h.s.* of Equation (10) is $\frac{1}{M_j g_2(\sigma_j^2,T_j)}\log\frac{n'}{\delta} \leq O(\frac{d}{\lambda}\log\frac{n'}{\delta})$. This implies $B \sim O(\frac{d}{\lambda}\log n')$ is sufficient.

□

8.1.4 Proof of Theorem 2

Proof. Take $m = \arg\max_{i=1,\ldots,n} c_i$. Since $w_t h_{jt} \geq 0, \forall j, t$, so $S = \sum_{i=1}^n c_i$.

Since $C = B$, not identifying maximum inner product is equivalent to index m not sampled within B samples. And we know that $P(\text{index } m \text{ sampled in a step}) = \frac{c_m}{S}$. Denote $A = \{\text{index m not sampled within B samples}\}$, then for any $\alpha \in (0, \frac{1}{2})$,

$$\begin{aligned} & P(\text{not identifying maximum inner product}) \\ = \;& E[\mathbb{1}(A)] = E[E[\mathbb{1}(A)|\frac{c_m}{S}]] \\ = \;& E[\left(1-\frac{c_m}{S}\right)^B] \leq E[exp\left(-B\frac{c_m}{S}\right)] \\ = \;& E[\exp\left(-\frac{Bc_m}{S}\right)\mathbb{1}(\frac{c_m}{S} > n^{-1}\alpha\log n)] \\ & +E[\exp\left(-\frac{Bc_m}{S}\right)\mathbb{1}(\frac{c_m}{S} \leq n^{-1}\alpha\log n)] \\ \leq \;& E[\exp\left(-Bn^{-1}\alpha\log n\right)] + P(\frac{c_m}{S} \leq n^{-1}\alpha\log n) \\ \leq \;& n^{-\rho} + P(\frac{c_m}{S} \leq n^{-1}\alpha\log n). \end{aligned} \quad (11)$$

For any $\alpha \in (0, \frac{1}{2})$ and $\forall \epsilon > 0$,

$$\begin{aligned} & P(\frac{c_m}{S} > n^{-1}\alpha\log n) = P(c_m > n^{-1}S\alpha\log n) \\ \geq \;& P(c_m > n^{-1}S\alpha\log n, n^{-1}S \leq \beta^{-1}+\epsilon) \\ = \;& P(c_m > n^{-1}S\alpha\log n | n^{-1}S \leq \beta^{-1}+\epsilon) \\ & P(n^{-1}S \leq \beta^{-1}+\epsilon) \\ \geq \;& P(c_m > (\beta^{-1}+\epsilon)\alpha\log n)P(n^{-1}S \leq \beta^{-1}+\epsilon) \end{aligned}$$

Since $c_j \overset{iid}{\sim} Exp(\beta)$, $E[S] = nE[c_j] = n\beta^{-1}$, $\text{Var}[S] = n\text{Var}[c_i] = \frac{n}{\beta^2}$. According to Chebyshev's Inequality, $\forall \epsilon > 0$,

$$\begin{aligned} & P(n^{-1}S \leq \beta^{-1}+\epsilon) = 1 - P(n^{-1}S - \beta^{-1} \geq \epsilon) \\ \geq \;& 1 - P(|n^{-1}S - \beta^{-1}| \geq \epsilon) \\ \geq \;& 1 - \frac{\text{Var}[n^{-1}S]}{\epsilon^2} = 1 - \frac{1}{n\beta^2\epsilon^2}. \end{aligned} \quad (12)$$

Since $c_j \overset{iid}{\sim} Exp(\beta)$, we have for $\alpha \in (0, \frac{1}{2})$,

$$\begin{aligned} & P(c_m > (\beta^{-1}+\epsilon)\alpha\log n) \\ = \;& 1 - P(c_m \leq (\beta^{-1}+\epsilon)\alpha\log n) \\ = \;& 1 - [1 - e^{-\beta(\beta^{-1}+\epsilon)\alpha\log n}]^n \\ = \;& 1 - (1 - \frac{1}{n^{(1+\beta\epsilon)\alpha}})^n \\ \sim \;& 1 - e^{-n^{1-(1+\beta\epsilon)\alpha}}. \end{aligned} \quad (13)$$

Since it holds for any $\epsilon > 0$, we can take $\epsilon = \beta^{-1}$, then from Equation (12, 13), we have

$$\begin{aligned} & P(\frac{c_m}{S} > n^{-1}\alpha\log n) \\ \geq \;& \left(1-\frac{1}{n\beta^2\epsilon^2}\right)\left(1-e^{-n^{1-(1+\beta\epsilon)\alpha}}\right) \\ = \;& (1-\frac{1}{n})(1-e^{-n^{1-2\alpha}}). \end{aligned} \quad (14)$$

From Equation (11), we know:

$$\begin{aligned} & P(\text{not identifying maximum inner product}) \\ \leq\ & n^{-\rho} + 1 - (1 - \frac{1}{n})(1 - e^{-n^{1-2\alpha}}) \\ \sim\ & O\left(\frac{1}{n^{\rho}}\right). \end{aligned}$$

□

8.1.5 Proof of Corollary 1

Proof. Assume $E(c_j) = \beta^{-1} > 0$ and $\text{Var}[c_j] = \gamma^{-2}$, then for any $\epsilon > 0$, the following equation still holds.

$$\begin{aligned} & P(n^{-1}S \leq \beta^{-1} + \epsilon) = 1 - P(n^{-1}S - \beta^{-1} \geq \epsilon) \\ \geq\ & 1 - P(|n^{-1}S - \beta^{-1}| \geq \epsilon) \\ \geq\ & 1 - \frac{\text{Var}[n^{-1}S]}{\epsilon^2} = 1 - \frac{1}{n\gamma^2\epsilon^2}. \end{aligned} \tag{15}$$

Take $\epsilon = \gamma^{-1}$ in the above equation, we have

$$P(n^{-1}S \leq \beta^{-1} + \epsilon) > 1 - \frac{1}{n}. \tag{16}$$

Since c_i is heavy-tailed, so for all $\mu > 0$, there exists $x_0 > 0$, such that for all $x \geq x_0$,

$$P(c_i \geq x) \geq e^{-\mu x}. \tag{17}$$

Take $\mu = \frac{1}{\beta^{-1}+\epsilon}$, when $(\beta^{-1} + \epsilon)\alpha \log n \geq x_0$, we have

$$\begin{aligned} & P(c_m > (\beta^{-1} + \epsilon)\alpha \log n) \\ =\ & 1 - P(c_m \leq (\beta^{-1} + \epsilon)\alpha \log n) \\ =\ & 1 - \left(P(c_i \leq (\beta^{-1} + \epsilon)\alpha \log n)\right)^n \\ \geq\ & 1 - \left(1 - e^{-\mu(\beta^{-1}+\epsilon)\alpha \log n}\right)^n \\ =\ & 1 - \left(1 - e^{-\alpha \log n}\right)^n \\ \sim\ & 1 - e^{-n^{1-\alpha}}. \end{aligned} \tag{18}$$

From Equation $(12, 16, 18)$, we know that

$$P(\frac{c_m}{S} > n^{-1}\alpha \log n) \geq (1 - \frac{1}{n})(1 - e^{-n^{1-\alpha}}).$$

From the proof of Theorem 2, we know the result of Corollary 1 holds. □

8.2 More comparison of experimental results

In order to get more informative comparisons, we summarize the results from Figure 3 in the following tables. Prec@1 and Prec@5 of MovieLens with $d = 50$ are shown in Table 1 and 2 respectively. From the results, we can see that Sampling-MIPS algorithm outperforms PCA-MIPS, LSH-MIPS and Diamond-MSIPS consistently on this dataset. Prec@1 and Prec@5 of Netflix with $d = 50$ are shown in Table 3 and 4 respectively. Note that the speedup of PCA-tree method depends on the depth of the PCA tree, which is corresponding to the number of candidates in each leaf node. A deeper PCA tree leads to a higher speedup with a tradeoff of a lower precision. The maximum depth in our experiment is too small to generate a point with a speedup greater than 5. This is the reason that in Table 3 and 4, there are no results shown for PCA-MIPS. But we can see from Figure 3 that with the current maximum depth chosen, the precision of PCA-MIPS is drastically reduced to almost 0 when speedup is less than 5. So with a bigger maximum depth, the result of PCA-MIPS will get even worst.

Table 1: Result of prec@1 for MovieLens $(d = 50)$

Speedup (prec@1)	5	10	20
Sampling-MIPS	99.95%	99.65%	65%
Diamond	68.35%	42.25%	25.85%
PCA	0.15%	0.00%	0.00%
LSH	4.75%	18.9%	0.05%

Table 2: Result of prec@5 for MovieLens $(d = 50)$

Speedup (prec@5)	5	10	20
Sampling-MIPS	87.38%	72%	13%
Diamond	11.65%	5.41%	1.52%
PCA	0.00%	0.00%	0.00%
LSH	9.73%	4.81%	0.00%

Table 3: Result of prec@1 for Netflix $(d = 50)$

Speedup (prec@1)	5	10	20
Sampling-MIPS	97.30%	77.15%	29.80%
Diamond	51.85%	31.4%	15.2%
PCA	NA	NA	NA
LSH	23%	1.05%	NA%

Table 4: Result of prec@5 for Netflix $(d = 50)$

Speedup (prec@5)	5	10	20
Sampling-MIPS	58.87%	28.7%	7.98%
Diamond	14.47%	7.26%	3.29%
PCA	NA	NA	NA
LSH	9.25%	0.29%	NA%

8.3 Extension to Maximum All-pair Dot-product (MAD) Search

8.3.1 Proposed algorithm for MAD Search

Our Sampling-MIPS algorithm can also be extended to solve the Maximum All-pair Dot-product (MAD) search problem. Instead of finding the maximum inner product for a specific query, MAD aims to find the maximum inner product over a set of m queries and n vectors in the database. More specifically, given two groups of d-dimensional vectors $\mathcal{W} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m\}$ and $\mathcal{H} = \{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n\}$, the goal of MAD problem is to find the index pairs $(i, j) \in \{1, \ldots, m\} \times \{1, \ldots, n\}$ who have the maximum or top-K dot products. When $m = 1$, MAD problem reduces to the MIPS problem. MAD is also used in many recommender systems when we aim to find the best $(\text{user}, \text{item})$ among all the possible pairs.

We can use the same idea in Sampling-MIPS to solve the MAD problem. The only difference is the sampling procedure. In MIPS problem, we first sample t, then sample j conditioned on t. In MAD problem, we simply add one more step. We still sample t first, but then we sample i and j independently. We use a m-by-d matrix $W = [\mathbf{w}_1 \ldots \mathbf{w}_m]^T$ to denote the set of m queries and use w_{it} to denote the t-th feature of the i-th query. Define $P(t|W) \propto \sum_{i=1}^m |w_{it}| \sum_{j=1}^n |h_{jt}|$, $P(i|t) = \frac{|w_{it}|}{\sum_{i=1}^m |w_{it}|}$, and $P(j|t) = \frac{|h_{jt}|}{\sum_{j=1}^n |h_{jt}|}$. We also assume that $\sum_{i=1}^m |w_{it}| \neq 0$ and $\sum_{j=1}^n |h_{jt}| \neq 0$ for all t. So we have

$$\begin{aligned} P(i, j|W) &= \sum_{t=1}^d P(i, j, t|W) \\ &= \sum_{t=1}^d P(i|t)P(j|t)P(t|W) \\ &\propto \sum_{t=1}^d |w_{it}h_{jt}|. \end{aligned}$$

Here, the distribution for sampling (i, j) is very similar to the distribution for sampling j in MIPS problem.

Similarly, alias table for sampling j can be constructed in pre-processing phase. In query-dependent phase, we only need to construct alias table for sampling t and i. Details are shown in Algorithm 5. The time complexity of each step of is also shown in Algorithm 5. The total time complexity is $O(md + B + Cd)$, while the naive time complexity is $O(mnd)$. We expect this algorithm to perform as well as it does in MIPS problem. The theoretical guarantee of this sampling MAD algorithm can also be proved in a similar manner as Sampling-MIPS.

Algorithm 5 Sampling-MAD

Pre-processing:

$s_t \leftarrow \sum_{j=1}^n |h_{jt}|, \forall t = 1, \cdots, d$

$k_t \leftarrow \sum_{i=1}^m |w_{it}|, \forall t = 1, \cdots, d$

Use alias table to construct $P(j|t) \leftarrow \text{multinomial}([|h_{1t}|, \ldots, |h_{nt}|]), \forall t$

Candidate Screening:

Use alias table to construct $P(t|W) \leftarrow \text{multinomial}([|k_1 s_1|, \ldots, |k_d s_d|])$ $P(i|t) \leftarrow \text{multinomial}([|w_{1t}|, \ldots, |w_{mt}|]), \forall t$ $\cdots O(md)$

$\mathbf{x} = [x_{11} \ldots, x_{mn}] \leftarrow [0, \ldots, 0]$

Specify sample size B

for $b = 1, \ldots, B$ **do** $\cdots O(B)$

  Use alias method to sample $t \sim P(t|W)$

  Use alias method to sample $i \sim P(i|t)$

  Use alias method to sample $j \sim P(j|t)$

  $x_{ij} \leftarrow x_{ij} + \text{sgn}(w_{it}h_{jt})$

end for

Prediction Phase:

Find the biggest C elements in $\mathbf{x}$, i.e., $|\mathcal{C}(W)| = C$ and $\mathcal{C}(W) \leftarrow \{(i, j)|x_{ij} \geq x_{i'j'}, \forall (i', j') \notin \mathcal{C}(W)\}$ $\cdots O(B)$

for $(i, j) \in \mathcal{C}(W)$ **do** $\cdots O(Cd)$

  Compute inner product $\mathbf{w}_i^T \mathbf{h}_j$

end for

Output: indexes of the top-K elements of $\{\mathbf{w}_i^T \mathbf{h}_j | (i, j) \in C(W)\}$ $\cdots O(C)$

8.3.2 Mathematical analysis of Sampling-MAD

We also have the similar theoretical results for MAD problem. We omit the proofs since they are very similar to Lemma 1, 2, Theorem 1, 2 and Corollary 1 for the MIPS case.

Lemma 3. *Define $x_{ij,b} = sgn(w_{it}h_{jt})$ if index pair (i, j, t) is sampled in the b-th sampling step, $x_{ij,b} = 0$ otherwise. Note that $x_{ij} = \sum_{b=1}^B x_{ij,b}$. Define*

$$S = \sum_{t=1}^d \sum_{i=1}^m \sum_{j=1}^n |w_{it}h_{jt}|,$$

then we have

$$E[x_{ij,b}] = \frac{\mathbf{w}_i^T \mathbf{h}_j}{S}$$

and

$$E[x_{ij}] = \frac{B\mathbf{w}_i^T \mathbf{h}_j}{S}.$$

We can then show the ranking of x_{ij}'s will be correct when we have enough samples.

Lemma 4. *Define $A_{ij} = \sum_{t=1}^d |w_{it}h_{jt}|$, then $E[(x_{ij,b} - x_{i'j',b})^2] = \frac{A_{ij} + A_{i'j'}}{S}, \forall (i, j) \neq (i', j')$. Therefore, from*

Lemma 3, we have

$$\begin{aligned}&Var(x_{ij,b} - x_{i'j',b})\\ =&\frac{A_{ij} + A_{i'j'}}{S} - \frac{(\mathbf{w}_i^T\mathbf{h}_j - \mathbf{w}_{i'}^T\mathbf{h}_{j'})^2}{S^2}\end{aligned}$$

and $Var(x_{ij} - x_{i'j'}) = BVar(x_{ij,b} - x_{i'j',b})$.

Theorem 3. *For some index pair* (I, J)*, define for* $(i, j) \in \{(i, j) : \mathbf{w}_i^T\mathbf{h}_j < \mathbf{w}_I^T\mathbf{h}_J\}$*:*

$$\begin{aligned}N &= \#\{(i, j) : \mathbf{w}_i^T\mathbf{h}_j < \mathbf{w}_I^T\mathbf{h}_J\} \text{ (assume } N \neq 0),\\ \Lambda_{ij} &= \frac{\mathbf{w}_I^T\mathbf{h}_J - \mathbf{w}_i^T\mathbf{h}_j}{S},\\ \Lambda &= \min_{\{(i,j):\mathbf{w}_i^T\mathbf{h}_j < \mathbf{w}_I^T\mathbf{h}_J\}} \Lambda_{ij},\\ T_{ij} &= \frac{(S + \mathbf{w}_I^T\mathbf{h}_J - \mathbf{w}_i^T\mathbf{h}_j)(\mathbf{w}_I^T\mathbf{h}_J - \mathbf{w}_i^T\mathbf{h}_j)}{S^2},\\ M_{ij} &= \frac{S^2}{(S + \mathbf{w}_I^T\mathbf{h}_J - \mathbf{w}_i^T\mathbf{h}_j)^2},\\ \sigma_{ij}^2 &= Var(x_{ij,b} - x_{IJ,b}).\end{aligned}$$

If sample size B *satisfies the following equation for all* $(i, j) \neq (I, J)$*,*

$$B \geq \frac{\log\frac{N}{\delta}}{M_{ij}\sigma_{ij}^2\left[(1 + \frac{T_{ij}}{\sigma_{ij}^2})\log(1 + \frac{T_{ij}}{\sigma_{ij}^2}) - \frac{T_{ij}}{\sigma_{ij}^2}\right]}, \qquad (19)$$

implying that $B \sim O(\frac{1}{\Lambda}\log(N))$*, then with error probability* $\delta \in (0, 1)$*, we have*

$$P(x_{IJ} > x_{ij}, \forall(i, j) \in \{\mathbf{w}_i^T\mathbf{h}_j < \mathbf{w}_I^T\mathbf{h}_J\}) \geq 1 - \delta.$$

Similar to MIPS problem, when $\mathbf{w}_I^T\mathbf{h}_J$ has the maximum inner product, N in the theorem above can be replaced by mn. That means that sample size $B = O(\frac{1}{\Lambda}\log(mn))$ is sufficient for identifying the maximum inner product in MAD problem and $\Lambda < 1$ here.

Theorem 4. *Assume* $w_{it}h_{jt} \geq 0$*, for all* $(i, j, t) \in \{1, \ldots, m\} \times \{1, \ldots, n\} \times \{1, \ldots, d\}$ *and* $\mathbf{w}_i^T\mathbf{h}_j \overset{iid}{\sim} Exp(\beta)$*. In Sampling-MAD algorithm, if we take* $C = B$*, where* C *means we calculate inner products of indexes with top-*C *scores, then when* $B \geq \frac{\rho}{\alpha}mn$*, where* $\alpha \in (0, \frac{1}{2})$ *and* $\rho \in (0, \frac{\alpha d(n-1)}{(d+1)n})$*,*

$$P(\text{not identifying maximum inner product}) \leq O\left(\frac{1}{(mn)^\rho}\right).$$

Therefore, the overall time complexity of Sampling-MAD is $O(md + B + Bd) = O(md + \frac{\rho}{\alpha}(d + 1)mn) < O(mnd)$.

Corollary 2. *Assume* $w_{it}h_{jt} \geq 0$*, for all* $(i, j, t) \in \{1, \ldots, m\} \times \{1, \ldots, n\} \times \{1, \ldots, d\}$*. If* $\mathcal{F}$ *is a continuous, non-negative, heavy-tailed distribution,* $\mathbf{w}_i^T\mathbf{h}_j \overset{iid}{\sim} \mathcal{F}$ *and* $E[(\mathbf{w}_i^T\mathbf{h}_j)^2] < \infty$*, then when* $B \geq \frac{\rho}{\alpha}mn$*, where* $\alpha \in (0, 1)$ *and* $\rho \in (0, \frac{\alpha d(n-1)}{(d+1)n})$*, take* $C = B$ *in Sampling-MIPS algorithm, we have*

$$P(\text{not identifying maximum inner product}) \leq O\left(\frac{1}{(mn)^\rho}\right).$$