Online Sketching Hashing

Cong Leng[1*] Jiaxiang Wu[1*] Jian Cheng[1] Xiao Bai[2] Hanqing Lu[1]

[1]National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences
[2]School of Computer Science and Engineering, Beihang University, China

{cong.leng, jiaxiang.wu, jcheng, luhq}@nlpr.ia.ac.cn

baixiao@buaa.edu.cn

Abstract

Recently, hashing based approximate nearest neighbor (ANN) search has attracted much attention. Extensive new algorithms have been developed and successfully applied to different applications. However, two critical problems are rarely mentioned. First, in real-world applications, the data often comes in a streaming fashion but most of existing hashing methods are batch based models. Second, when the dataset becomes huge, it is almost impossible to load all the data into memory to train hashing models. In this paper, we propose a novel approach to handle these two problems simultaneously based on the idea of data sketching. A sketch of one dataset preserves its major characters but with significantly smaller size. With a small size sketch, our method can learn hash functions in an online fashion, while needs rather low computational complexity and storage space. Extensive experiments on two large scale benchmarks and one synthetic dataset demonstrate the efficacy of the proposed method.

1. Introduction

With the rapid advance of digital devices and Internet, we are entering the age of big data. The ubiquitous large scale datasets pose significant challenges to the traditional machine learning and computer vision. For example, nearest neighbors (NN) search is a critical component in many learning algorithms such as clustering, retrieval and matching [3]. However, as the dataset becomes huge, exhaustive NN search is prohibitive because of the high complexity in both time and storage. As an alternative, hashing based approximate nearest neighbors (ANN) search has attracted much attention recently, owing to its efficiency in both search speed and storage.

In hashing based ANN approaches, data in the original space is embedded to binary codes in the Hamming space with hash functions. The goal is to approximate the input distance with the Hamming distance which can be calculated extremely fast with XOR operation in modern CPU. Early works, such as Locality Sensitive Hashing (LSH) [14, 2], construct hash functions based on random projection. These data-independent methods often require long code length to achieve acceptable search accuracy. To overcome this problem, recent research attentions have been shifted to data-dependent hashing techniques. Representative works include Iterative Quantization (ITQ) [9], Isotropic Hashing (IsoH) [17], Spherical Hashing (SpH) [12] and so on [25, 19, 24]. In these methods, rather than randomly generated, hash functions are learned from data distribution or supervised information. The learning process is driven by a core principle, *i.e.*, similar samples in the original space should have close codes. Although promising results have been obtained by these hashing methods for ANN search, two critical problems are seldom mentioned.

First, in many real-world applications, data becomes available continuously in streaming fashion. For example, a search engine company like Google has numerous new web pages and images continuously arriving at the data centers every day. In such environments, the queries must be answered continuously, based on the total data that has arrived so far. However, most of the existing hashing models are based on a batch learning fashion. That is to say, when new data arrives, they have to accumulate all the available data and re-train new hash functions, which is apparently a less efficient learning manner for streaming data. This problem poses us the first challenge, *i.e.*, how to learn from the newly arrived data which is available after the hash functions have already been generated from previous dataset.

Second, for truly large scale datasets, data is usually stored on a distributed disk group and is too large to be read into memory. Actually, it is even not possible to load the data crawled in one day by Google into memory, let alone all the data. Moreover, one processor is often incapable of han-

*These authors contributed equally to this work.

dling the large scale datasets in a feasible amount of time. For example, for data matrix $X \in \mathbb{R}^{d\times n}$, its PCA transform needs a time complexity of $O(nd^2+d^3)$ which is infeasible when n and d are too large. This poses us the second challenge, *i.e.*, how to overcome the scalability problem in the hash function learning process.

More generally, the first challenge mentioned above is targeted by online learning. The goal of online learning is to produce a sequence of hypotheses where the current hypothesis describes all data available so far, but depends on only the current training data [30]. Extensive algorithms and architectures about this topic have been proposed [29, 23, 10, 4, 20], but seldom been introduced to hash function learning. To the best of our knowledge, the only work which attempted to learn hash function online was proposed in [13].

In this paper, we propose a novel online hashing approach to address the two problems mentioned above simultaneously. The proposed method, named as online sketching hashing (OSH), is of online learning fashion, and allows a dramatic reduction in computation and storage overhead. Our work is largely inspired by the idea of "data sketching" [21, 18]. A sketch of one dataset preserves the main property of interest but with a significantly smaller size, such that computations can be performed on the sketch rather than the whole dataset without much loss of information [21]. Our approach maintains a small size sketch for the streaming data online, and we demonstrate that the hash functions can be efficiently learned based on this sketch. To overcome the mean-varying problem in online hash function learning, we propose a new sketching algorithm for zero-meaned stream data. Extensive experiments demonstrate that our method can keep the performance comparable to batch learning based methods even when only a very small sketch is maintained, and outperform existing online hashing method in both accuracy and efficiency.

2. Related Work

In this section, we briefly review some representative works about online learning and online hashing.

2.1. Online Learning

Although rarely discussed in the area of hashing, online learning has attracted much attention in machine learning community. For example, an online version of Bagging and Boosting was proposed in [29] to adapt the original AdaBoost and Bagging algorithms for learning from the stream data. The online Boosting [29] was further applied to obtain an online feature selection approach [10]. In [32], the authors proposed an online learning strategy for SVM, in which only the support vectors are preserved in each online step and added into the training set for next step.

A family of online learning methods for regression, binary classification, uniclass prediction, multiclass prediction and sequence prediction tasks were presented in [4] based on the "Passive-Aggressive (PA)" idea. Taking the binary classification as an example, the general concept of PA is that upon arrival of a new example, the new classifier should be constructed in such a way that it remains as close as possible to the old one (*i.e.* passive), while at the same time minimizes the loss on the new example (*i.e.* aggressive). Formally, assuming each data x is associated with a label y, let $\langle x_t, y_t\rangle$ denote the labeled example presented to algorithm at round *t*. The classifier used in the algorithm on last round is denoted as w_{t-1}, then the objective of PA algorithm can be formulated as:

$$\begin{aligned} w^t &= \arg\min_{w} \frac{1}{2}\|w - w^{t-1}\|^2 + C\xi \\ \text{s.t.} \quad & L(w; \langle x^t, y^t\rangle) \le \xi \quad \text{and} \quad \xi \ge 0 \end{aligned} \tag{1}$$

where $L(w; \langle x, y\rangle)$ is some kind of predefined loss function, *e.g.* hinge loss, on one example. C is a positive parameter which controls the influence of the "aggressive" term.

2.2. Online Hashing

To the best of our knowledge, online kernel-based hashing (OKH) [13] is the only work which attempts to learn hash functions online. Inspired by [4], the proposed OKH adapts the hash function accommodate to each new pair of data along the line of "Passive-Aggressive" method [4]. Given a new pair of data (x_i^t, x_j^t) and their similarity label $s_{ij} \in \{-1, +1\}$ at round t, OKH updates the hash function so that it can fit the newly available data pair and meanwhile coincide to the old one. Formally, denoting the hash function learned on last round as W^{t-1} (a matrix), the objective of online hashing in [13] can be formulated as:

$$\begin{aligned} W^t &= \arg\min_{W} \frac{1}{2}\|W - W^{t-1}\|_F^2 + C\xi \\ \text{s.t.} \quad & L(W; \langle (x_i^t, x_j^t), s_{ij}\rangle) \le \xi \ \text{ and } \ \xi \ge 0 \end{aligned} \tag{2}$$

where $L(W; \langle (x_i, x_j), s_{ij}\rangle)$ is the predefined loss function defined on a pair of labeled data and C is a control parameter. Comparing Eq.(2) with Eq.(1), it is easy to find they are very similar. OKH extends the PA framework into online hashing with a different definition of loss function L.

The OKH algorithm deals with only a pair of new available samples at a time. In the worst case, the hash functions will be updated once for every pair of new samples, which is often less efficient in practice. Furthermore, it assumes the availability of a similarity label for the new data pair, which is often expensive to obtain in practice.

3. The Proposed Method

In this work, we assume that the data is available in a stream form. Let $\mathcal{D}_i$ denote the data chunk received

at round i, where $i = \{1, 2, \cdots\}$. In particular, $\mathcal{D}_i = [x_1^i, x_2^i, \cdots, x_{m_i}^i] \in \mathbb{R}^{d \times m_i}$ contains m_i samples, and each $x_j \in \mathbb{R}^d$ is a data of d dimension. The mean of the data chunk $\mathcal{D}_i$ is denoted by $\overline{\mathcal{D}_i}$. Let X_t denote the data matrix accumulated from round 1 to round t, namely, $X_t = [\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_t]$. The size of X_t is $d \times n_t$ where $n_t = \sum_{i=1}^{t} m_i$ is the number of data available till round t. μ_t is the mean of data in X_t. $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. $\|\cdot\|_2$ denotes the spectral norm of a matrix. One highlight of online learning is that when learning new information at round t, the algorithm should not access the previously seen data $\mathcal{D}_1, \cdots, \mathcal{D}_{t-1}$.

3.1. Preliminary

Putting aside for the moment the problem that data comes as a stream, suppose we have a $d \times n$ matrix $X = [x_1, x_2, \cdots, x_n]$ where each column x_i is a sample in the dataset. Let $H = [h_1, h_2, \ldots, h_r]$ denote a sequence of r hash functions. For simplicity, following many other hashing works [2, 33, 9, 17, 27], we use linear projection coupled with mean thresholding as a hash function. Specifically, given a projection vector $w_k \in \mathbb{R}^d$, the k^{th} hash function is defined as:

$$h_k(x) = \text{sgn}(w_k^T x + b_k) \tag{3}$$

where b_k is the negative mean of the projected data, *i.e.* $b_k = -\frac{1}{n}\sum_{j=1}^{n} w_k^T x_j$, and therefore the hash function can be rewritten as:

$$h_k(x) = \text{sgn}(w_k^T (x - \mu)) \tag{4}$$

where $\mu = \frac{1}{n}\sum_{j=1}^{n} x_j$ is the mean of all the data.

For the binary codes to be efficient, as indicated in [34], typically two requirements should be satisfied: (1) each bit has a 50% chance to be +1 or -1, *i.e.* $\sum_i h_k(x_i) = 0, k = 1, 2, \cdots, r$; (2) different bits are independent of each other. For the first requirement, Wang *et al.* [33] have proved that the constraint $\sum_i h_k(x_i) = 0$ is equivalent to maximizing the variance for the k-th bit. The second "independent" requirement is relaxed to the pairwise decorrelation of bits in [34], and further relaxed to the orthogonality constraints of the projection directions $\{w_1, w_2, \cdots, w_r\}$ in [33]. Denote $W = [w_1, w_2, \cdots, w_r] \in \mathbb{R}^{d \times r}$, by dropping the non-differentiable sgn$(\cdot)$ function, then the objective can be formally written in a matrix form as:

$$\begin{aligned} \max_{W \in \mathbb{R}^{d \times r}} \quad & tr(W^T (X - \mu)(X - \mu)^T W) \\ \text{s.t.} \quad & W^T W = I_r \end{aligned} \tag{5}$$

where the notation $(X - \mu)$ means the matrix $[x_1 - \mu, x_2 - \mu, \cdots, x_n - \mu]$, which is equivalent to centering the data. This objective function is exactly the same as that of Principal Component Analysis (PCA). The optimized projection

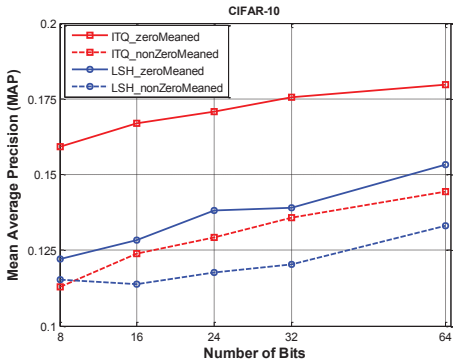

Figure 1. The mean average precision of LSH [2] and ITQ [9] with Hamming ranking on CIFAR-10 dataset. Two versions of the algorithm are reported. In the first version (dashed line), the data is not zero-meaned before training. In the second version (solid line), data is zero-meaned. The two versions share completely the same parameters in the experiments.

W can be obtained by solving the top r eigenvectors corresponding to the r biggest eigenvalues of the data covariance matrix[1] $(X - \mu)(X - \mu)^T$. For convenience, we denote the covariance matrix of X as $cov(X)$ below. Nevertheless, directly using the optimized W as hash projections will result in the unbalance problem, *i.e.*, most of the information is included in the top eigenvectors [17]. We leave this problem to detail and handle it in Section 3.2.

In many existing hashing works, the data is first normalized to have zero mean before training [33, 9], or assumed to be inherently zero-centered, then $b_k = 0$ for any hash function $h_k(x)$. The objective comes to a slighter form:

$$\max_{W \in \mathbb{R}^{d \times r}} \quad tr(W^T X X^T W) \quad \text{s.t.} \quad W^T W = I_r \tag{6}$$

It is worth noting that the zero-mean normalization step will largely influence the performance of hashing algorithms. As can be seen in Fig.1, with a zero-mean normalization to the data, whether the data-independent method LSH [2] or a more sophisticated data-dependent method like ITQ [9] will get a substantial improvement over the version in which data is not normalized. As we will find in the next subsection, the zero mean property is critical when it comes to designing an online hashing method based on data sketching.

3.2. Online Hashing via Data Sketching

The model in Eq.(5) described above is a fairly fundamental and effective one among the data-dependent hashing methods. However, two major limitations inhibit its use in practical applications: (1) It is obviously a batch based

[1] This is a slight abuse of notation. In statistics covariance matrix is defined as $\frac{1}{n-1}(X - \mu)(X - \mu)^T$, but in this paper we just ignore the scale factor $\frac{1}{n-1}$ and define the matrix $(X - \mu)(X - \mu)^T$ as covariance matrix in order to simplify the presentation.

learning method. Upon the arrival of new data chunk, the model has to accumulate it with the old ones and then retrain the hash functions. (2) When the data size n and data dimension d are too large, it is prohibitive to load all the data into memory. It is also infeasible to calculate the covariance matrix $cov(X)$ and its eigen-decomposition. Next we propose a novel approach to overcome these two limitations simultaneously based on the idea of "data sketching".

3.2.1 Online Data Sketching

Overall speaking, a sketch of matrix P is another matrix Q which is much smaller than P, but still preserves the properties of interest. In this way, the storage of the sketch Q will be much easier, and the computations can be performed much faster than with the original P [18, 8, 21]. Formally, given a matrix $P \in \mathbb{R}^{d\times n}$, we aim to maintain a much smaller matrix $Q \in \mathbb{R}^{d\times l}$ with $l \ll n$ as an approximation to P. The goal is to track an ε-approximation to the norm of matrix along any direction, *i.e.*,

$$\forall x,\ \|x\| = 1 \qquad |\|P^T x\|^2 - \|Q^T x\|^2| \le \varepsilon \|P\|_F^2 \quad (7)$$

Since $\|x\| = 1$, it can be viewed as a direction in the space. This guarantees that in any direction x, P and Q are close, where the closeness is defined by the Frobenius norm of P.

Owing to its importance, the data sketching problem has been carefully investigated in the literature. An extensive body of algorithms exist, including CUR decomposition [5], random projection [31, 22], column sampling method [7, 1], and Nyström method [6]. However, most of these methods are not streaming algorithms because they need several passes over the input matrix. The latest significant effort is represented by Frequent Directions (FD) proposed by Liberty [21]. Inspired by the works in finding frequent items, Liberty investigated how to apply the Misra-Gries technique [26] to matrix sketching.

The FD method can be summarized in Algorithm 1. The input to the algorithm is a $d\times n$ data matrix P and sketching size l. Each column P_i of matrix P represents a sample in the dataset. Columns from P will replace the all zero valued columns in Q. Once there is no zero valued column in Q, half of columns in the sketch will be emptied with two steps. First, the sketch is rotated from right with the SVD decomposition of Q so that its columns are orthogonal and in descending magnitude order.[2] Note that this step does not lose anything since for $C = US$ we have $QQ^T = CC^T$. In the second step, the Misra-Gries technique [26] is used to reduce half of singular values in S to be zero. Accordingly, the right half of columns in $U\widehat{S}$ will be all zeros.

Algorithm 1 Frequent Directions (Liberty [21])

Input: Data matrix $P \in \mathbb{R}^{d\times n}$, sketch matrix $Q \in \mathbb{R}^{d\times l}$.
Output: Sketch matrix Q.

```
if Q not exists then
    Q ⟵ all zeros d × l matrix
end if
for each column P_i in P, do
    Insert P_i into a zero valued column of Q
    if Q has no zero valued columns then
        [U, S, V] = SVD(Q)
        \\ C = US    [just for notation]
        Set δ = s_{l/2}^2    [the squared (l/2)^th entry of S]
        Set Ŝ = sqrt(max(S^2 − I_l δ, 0))
        Q = UŜ
    end if
end for
```

Despite the algorithm's apparent simplicity, it needs a great of effort to provide tight bounds for its performance [21]. Formally, we have the following lemma:

Lemma 1. (Liberty [21]) Apply Algorithm 1 to matrix P to obtain a sketch Q with prescribed l, then

$$\forall x,\ \|x\| = 1 \qquad 0 \le \|P^T x\|^2 - \|Q^T x\|^2 \le \frac{2}{l}\|P\|_F^2 \quad (8)$$

or

$$0 \le \|PP^T - QQ^T\|_2 \le \frac{2}{l}\|P\|_F^2 \quad (9)$$

Proof. *The proof can be referred in* [21].

Note that the FD algorithm works in a streaming fashion as only one pass to the data is required. In other words, the input matrix P can be a streaming matrix for a stream of data. This property, we believe, can be very useful for the online learning. However, specific to our hashing problem, it will encounter some difficulties by directly applying the algorithm, as we will see in the next section.

3.2.2 Zero Mean Sketching

In this paper, we attempt to employ the favorable property that $PP^T \approx QQ^T$ in [21] to handle the scalability and streaming data issue in hashing. A very straightforward way is sketching the matrix $X - \mu$ in Eq.(5) with Algorithm 1 so that we can get a significantly smaller sketch Y which approximates $X - \mu$ well with

$$YY^T \approx (X-\mu)(X-\mu)^T \quad (10)$$

However, this simple method is infeasible for online hashing. This is because the data is continuously changing, and therefore the mean of data μ changes too. When a

[2] In [21], the sketch size l is implicitly assumed to be smaller the data dimension d. Therefore, the SVD decomposition of Q will have the form as $Q = USV^T$ where $U \in \mathbb{R}^{d\times l}, V \in \mathbb{R}^{l\times l}$ and S is a $l \times l$ diagonal matrix. Obviously, $U^T U = V^T V = VV^T = I_l$ and so that V can be seen as rotation matrix. Besides, $l/2$ is assumed to be an integer. In this paper, we follow these assumptions as in [21] in the next sections.

new data chunk $\mathcal{D}_t$ arrives at round t, since the mean of the dataset changes to μ_t, we need to re-sketch all the data $[\mathcal{D}_1 - \mu_t, \mathcal{D}_2 - \mu_t, \cdots, \mathcal{D}_t - \mu_t]$.

To give a more intuitive understanding about this problem, let's consider a simple example. At the first round, we have a data chunk $\mathcal{D}_1$ with mean as $\mu_1 = \overline{\mathcal{D}_1}$, then the zero-meaned data at this time is $\mathcal{D}_1 - \mu_1$. After sketching this zero-meaned matrix into a much smaller one as Y_1 with Algorithm 1, we get the next chunk $\mathcal{D}_2$. At this time, the mean of the dataset will change to $\mu_2 = \frac{m_1\overline{\mathcal{D}_1}+m_2\overline{\mathcal{D}_2}}{m_1+m_2}$. Obviously, $\mu_2 \neq \mu_1$ and the zero-meaned data will change to $[\mathcal{D}_1 - \mu_2, \mathcal{D}_2 - \mu_2]$. Although we have already sketched $\mathcal{D}_1 - \mu_1$ with Y_1, it is useless now because the mean of data has changed. We have to re-sketch the matrix $[\mathcal{D}_1 - \mu_2, \mathcal{D}_2 - \mu_2]$. That is to say, the algorithm needs to have access to $\mathcal{D}_1$ again at the second round, which does not meet the requirement of online learning.

We now show how this problem can be overcome. Recall that $X_t = [\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_t]$ denotes the data matrix accumulated from round 1 to round t, μ_t is the mean of the data in X_t and n_t is the data size. Intuitively, in order to avoid the mean-varying problem, one can augment every data chunk with a virtual sample [23], which is carefully chosen to correct the time-varying mean. Specifically, for a stream X_t we design a matrix E_t as:

$$\begin{aligned} E_t = [\mathcal{D}_1 - \overline{\mathcal{D}_1}, \quad & \mathcal{D}_2 - \overline{\mathcal{D}_2}, \sqrt{\frac{n_1 m_2}{n_1+m_2}}(\overline{\mathcal{D}_2} - \mu_1), \cdots, \\ & \mathcal{D}_i - \overline{\mathcal{D}_i}, \sqrt{\frac{n_{i-1} m_i}{n_{i-1}+m_i}}(\overline{\mathcal{D}_i} - \mu_{i-1}), \cdots, \\ & \mathcal{D}_t - \overline{\mathcal{D}_t}, \sqrt{\frac{n_{t-1} m_t}{n_{t-1}+m_t}}(\overline{\mathcal{D}_t} - \mu_{t-1})] \end{aligned}$$

or for simplicity let

$$\begin{aligned} \widehat{\mathcal{D}_1} &= \mathcal{D}_1 - \overline{\mathcal{D}_1}, \\ \widehat{\mathcal{D}_i} &= [\mathcal{D}_i - \overline{\mathcal{D}_i}, \sqrt{\frac{n_{i-1} m_i}{n_{i-1}+m_i}}(\overline{\mathcal{D}_i} - \mu_{i-1})] \end{aligned} \tag{11}$$

then

$$E_t = [\widehat{\mathcal{D}_1}, \widehat{\mathcal{D}_2}, \cdots, \widehat{\mathcal{D}_t}] \tag{12}$$

The essence of the designing of $\widehat{\mathcal{D}_i}$ is to augment the zero-meaned data chunk $\mathcal{D}_i - \overline{\mathcal{D}_i}$ with an additional vector $\sqrt{\frac{n_{i-1}m_i}{n_{i-1}+m_i}}(\overline{\mathcal{D}_i} - \mu_{i-1})$ which can be viewed as a virtual sample. Also note that each component $\widehat{\mathcal{D}_i}$ in E_t depends on the mean of previous chunks which can be easily maintained and updated with

$$\mu_i = \frac{n_{i-1}\mu_{i-1}}{n_{i-1}+m_i} + \frac{m_i\overline{\mathcal{D}_i}}{n_{i-1}+m_i} \tag{13}$$

Lemma 2. For any $i \geq 2$, we have

$$cov(X_i) = cov(X_{i-1}) + \widehat{\mathcal{D}_i}\widehat{\mathcal{D}_i}^T \tag{14}$$

Algorithm 2 Zero Mean Sketching

Input: Streaming data chunk $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_k$,
All zeros matrix Y of size $d \times l$.

1: Sketch $\mathcal{D}_1 - \overline{\mathcal{D}_1}$ into Y with Algorithm 1
2: $n \leftarrow m_1$ and $\mu \leftarrow \overline{\mathcal{D}_1}$
3: **for** $i = 2 : k$, **do**
4: Sketch $[\mathcal{D}_i - \overline{\mathcal{D}_i}, \sqrt{\frac{nm_i}{n+m_i}}(\overline{\mathcal{D}_i} - \mu)]$ into Y
5: $\mu \leftarrow \frac{n\mu}{n+m_i} + \frac{m_i\overline{\mathcal{D}_i}}{n+m_i}$ [update the data mean]
6: $n \leftarrow n + m_i$ [update the data size]
7: **end for**

Proof Sketch. It is easy to find that

$$cov(X_i) = \underbrace{(X_{i-1} - \mu_i)(X_{i-1} - \mu_i)^T}_{\text{term I}} + \underbrace{(\mathcal{D}_i - \mu_i)(\mathcal{D}_i - \mu_i)^T}_{\text{term II}}$$

With some algebraic manipulation, we have

$$\begin{aligned} \text{term I} &= cov(X_{i-1}) + n_{i-1}(\mu_{i-1} - \mu_i)(\mu_{i-1} - \mu_i)^T \\ \text{term II} &= (\mathcal{D}_i - \overline{\mathcal{D}_i})(\mathcal{D}_i - \overline{\mathcal{D}_i})^T + m_i(\overline{\mathcal{D}_i} - \mu_i)(\overline{\mathcal{D}_i} - \mu_i)^T \end{aligned}$$

Plugging Eq.(13) into both terms will lead to the equation in Eq.(14). □

Moreover, with the definition of E_t in Eq.(12) and Lemma 2, we come to the following proposition:

Theorem 3. For any round t, we have

$$E_t E_t^T = cov(X_t) \tag{15}$$

Proof. Because $\widehat{\mathcal{D}_1}\widehat{\mathcal{D}_1}^T = cov(X_1)$, we have

$$\begin{aligned} E_t E_t^T &= \widehat{\mathcal{D}_1}\widehat{\mathcal{D}_1}^T + \widehat{\mathcal{D}_2}\widehat{\mathcal{D}_2}^T + \widehat{\mathcal{D}_3}\widehat{\mathcal{D}_3}^T + \cdots + \widehat{\mathcal{D}_t}\widehat{\mathcal{D}_t}^T \\ &= cov(X_1) + \widehat{\mathcal{D}_2}\widehat{\mathcal{D}_2}^T + \widehat{\mathcal{D}_3}\widehat{\mathcal{D}_3}^T + \cdots + \widehat{\mathcal{D}_t}\widehat{\mathcal{D}_t}^T \\ &= cov(X_2) + \widehat{\mathcal{D}_3}\widehat{\mathcal{D}_3}^T + \cdots + \widehat{\mathcal{D}_t}\widehat{\mathcal{D}_t}^T \quad (Lemma\ 2) \\ &= cov(X_3) + \cdots + \widehat{\mathcal{D}_t}\widehat{\mathcal{D}_t}^T \qquad \cdots\cdots \\ &= cov(X_{t-1}) + \widehat{\mathcal{D}_t}\widehat{\mathcal{D}_t}^T \\ &= cov(X_t) \end{aligned}$$

This completes the proof. □

Theorem 3 implies that if we can find a sketch Y_t which approximates E_t well with $Y_t Y_t^T \approx E_t E_t^T$ at any round t, then we can also obtain a proper approximation to the covariance matrix of X_t since $E_t E_t^T = cov(X_t)$. This is exactly what we want because, as we have demonstrated, approximating the covariance matrix of X_t online is difficult due to the mean-varying problem. Based on these results, we propose a new sketching method for hashing in Algorithm 2. Our algorithm works in a streaming fashion. Now we give a relative bound for the approximation of $cov(X_t)$.

Theorem 4. If Y_t is the result of applying Algorithm 2 to the stream data X_t with prescribed sketch size l, then:

$$0 \leq \|cov(X_t) - Y_t Y_t^T\|_2 \leq 2\|X_t - \mu_t\|_F^2/l \qquad (16)$$

Proof. As we have proved above, $cov(X_t) = E_t E_t^T$, then according to Lemma 1,

$$0 \leq \|cov(X_t) - Y_t Y_t^T\|_2 \leq 2\|E_t\|_F^2/l \qquad (17)$$

On the other hand, with some simple derivation, one can show that

$$\begin{aligned} \|E_t\|_F^2 &= trace(E_t E_t^T) \\ &= trace(cov(X_t)) \\ &= trace((X_t - \mu_t)(X_t - \mu_t)^T) \\ &= \|X_t - \mu_t\|_F^2 \end{aligned} \qquad (18)$$

Combining the results in Eq.(17) and Eq.(18) completes the proof. □

3.2.3 Online Hash Function Learning

After applying Algorithm 2 to the streaming data X_t, we can obtain a sketch $Y_t \in \mathbb{R}^{d\times l}$ with $l \ll n$ at round t so that $Y_t Y_t \approx cov(X_t)$. Accordingly, the objective in Eq.(5) can be rewritten as

$$\max_{W_t \in \mathbb{R}^{d\times r}} \quad tr(W_t^T Y_t Y_t^T W_t) \quad \text{s.t.} \quad W_t^T W_t = I_r \qquad (19)$$

where W_t is the hash projections at round t. The optimized W_t can be obtained by concatenating top r eigenvectors of $Y_t Y_t^T$ (*i.e.* top r left singular vectors of Y_t). However, the eigen-decomposition of $Y_t Y_t^T$ leads to a time complexity of $O(d^3)$, which is prohibitive when d is very large.

In this work, we follow the assumption in [21] that $l \leq d$. As we will find in the experiments, typically $l = 200$ is sufficient to achieve comparable results. When d is too large ($d \gg l$), Y_t will be a very "tall" matrix. For a matrix of this form, an efficient variant of SVD decomposition exists. The variant, denoted as RSVD, includes three major steps [15]:

(1) Compute the QR decomposition of $Y_t \in \mathbb{R}^{d\times l}$ as $Y_t = QR$, in which the columns of $Q \in \mathbb{R}^{d\times l}$ are orthogonal and $R \in \mathbb{R}^{l\times l}$ is an upper triangular matrix (with $O(dl^2)$ complexity).

(2) Compute the SVD of R as $R = U_1 S V^T$ (with $O(l^3)$ complexity).

(3) Compute $U = QU_1$ then U is the union of eigenvectors of $Y_t Y_t^T$ (with $O(dl^2)$ complexity).

Specific to our problem, the optimized W_t is just the left r columns of U. The time complexity of RSVD is $O(dl^2+l^3)$, which is much more efficient than original $O(d^3)$ as $d \gg l$.

On the other hand, it will lead to the unbalance problem if the optimized W_t is directly used to be hash projections [17, 9]. Most of the information is contained in the most significant eigenvectors while the remainders are often noisy. This can be relieved by applying a rotation to W_t. For example, Gong *et al.* learned a rotation to minimize the quantization error [9]. In this paper, we apply a random rotation $R \in \mathbb{R}^{r\times r}$ to the learned W_t so that the hash projections become $W_t R$. One fact motivates us to make this choice, that is, in order to learn a rotation the algorithm like ITQ [9] needs the whole original training data but in our online learning setting only a sketch exists. Besides, the learning-free random rotation is more efficient and favorable to multi-table lookup as in LSH.

3.3. Complexity Analysis

Here we analyze the complexity of our online sketching hashing. Assuming there is a stream of data chunk $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_t$, we need to learn new hash functions at every round $i = 1, 2, \cdots, t$. We analyze the accumulated time complexity and space complexity of hash function learning from round 1 to round t.

Time complexity: Since the sketching step is in a stream fashion, as indicated in [21], the overall time complexity of this step is $O(n_t dl)$, where n_t is the number of accumulated data at round t. For each round, the complexity of learning W_i is the same as $O(dl^2 + l^3)$. Therefore, the overall accumulated time complexity is $O(n_t dl + tdl^2 + tl^3)$.

Space complexity: In our work, all the operations are conducted on a $d \times l$ sketch of the data, space overhead of which is $O(ld)$. The RSVD decomposition step occupies a space of $O(ld + l^2)$ and we need $O(dr)$ space to store the learned hash projections. The overall space complexity is $O(ld + l^2 + dr)$.

4. Experiments

In this section, three sets of experiments are conducted to verify three issues of our online sketching hashing (OSH) algorithm as follows: (1) To verify if enough information has been preserved in the sketch, the proposed OSH algorithm is compared to the state-of-the-art batch based hashing algorithms. (2) To verify the effectiveness and efficiency of OSH under online setting, it is compared to the existing online hashing algorithm, *i.e.*, online kernel-based hashing (OKH) [13]. (3) To evaluate the scalability of OSH, experiments are conducted on a large scale synthetic dataset.

4.1. Experimental Setting

Datasets: Two large scale benchmarks are employed to evaluate the proposed method: CIFAR-10[3] and GIST-1M[4].

[3] http://www.cs.toronto.edu/~kriz/cifar.html

[4] http://corpus-texmex.irisa.fr/

CIFAR-10 consists of 60K 32×32 color images in 10 classes, with 6,000 images per class. We extract 512 dimensional GIST descriptor [28] to represent each image. GIST-1M contains one million 960 dimensional GIST descriptors extracted from random images. For both datasets, we randomly select 1,000 data points as queries and use the rest as database and training set. For CIFAR-10 dataset, since every image in this dataset is assigned a class label, the ground truth is defined as semantic neighbors based on label agreement. Because there is no semantic ground truth provided for the GIST-1M dataset, following the same setting as in [24, 33], top 2 percentile nearest neighbors in Euclidean space are taken as ground truth.

Evaluation Criterions: To perform fair evaluation, we follow the Hamming ranking search strategy commonly used in the literature. The retrieval performance is evaluated with mean average precision (MAP) which is one of the most comprehensive criterion in retrieval. All the experiments are conducted in an unsupervised setting. For OKH, we follow the setting in [13] that two samples are considered to be similar if their Euclidean distance is within top 5% of the whole dataset. All the reported results are averaged over 5 independent runs. Our method is implemented with MATLAB and all the experiments are run on an Intel i7-2600 @ 3.4GHz CPU with 20GB RAM.

4.2. Comparison to Batch Based Methods

In the first experiment, we want to ensure that the sketching step will not lose too much information necessary for hash function learning. To verify this, we compare our method with the batch learning based methods such as ITQ [9], IsoH [17], AGH [25], SpH [12], KMH [11] and RMMH [16]. Among them, ITQ, IsoH and our OSH share the same hashing model in Section 3.1. In this experiment, we assume that all the training data is available and zero-centered before training. For the proposed OSH, we first find a small sketch for the full training set and then apply the learning method in Section 3.2.3 to obtain hash functions. We empirically set the sketch size l to be $l = 200$ for both datasets. All the source codes of the baseline methods are provided by the authors.

Fig.2 shows the MAP scores of all algorithms using Hamming ranking on two datasets with different code lengthes. We can clearly observe that the proposed OSH achieves comparable accuracy with ITQ and IsoH, and outperforms other baselines on both datasets. The performance of OSH is slightly inferior to the best competitor ITQ but still acceptable in consideration of the training is only based on a sketch of size 200. In fact, on GIST-1M the MAP scores of ITQ, IsoH and OSH are close to each other, especially when the code size increases. All these results suggest that the sketching step does not lose much information needed for learning hash functions. In addition, the

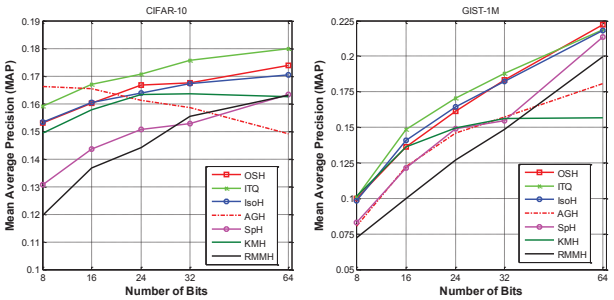

Figure 2. Hamming ranking performance: mean average precision (MAP) of different algorithms with different code lengthes on (a) CIFAR-10 and (b) GIST-1M. (Best viewed in color)

rotation matrix to the PCA projections in ITQ and IsoH is learned from the data, while in our method, only a random rotation is used. The results in Fig.2 demonstrate that random rotation is often adequate to balance the information contained in different PCA projections, which was also observed in [9].

4.3. Comparison to Online Hashing

For the second experiment, we focus on learning hash functions online for CIFAR-10 and GIST-1M. In the online setting, the training data is divided into multiple data chunks, which become available as a stream, to simulate the real-world situations. We compare the proposed method with OKH [13], which is the only existing online hashing method thus far. The source code of OKH is provided by the authors. LSH [2] is also chosen as a baseline, since its data-independent hash functions can be easily fitted to online setting. For both datasets, the training data is evenly divided into 100 chunks (*e.g.* 100 rounds). For CIFAR-10, we evaluate the MAP scores after each round. However, this becomes too time consuming for GIST-1M, thus we only evaluate the performance selectively on this dataset.

Fig.3 shows the MAP scores at different rounds on two datasets with 16, 32 and 64 bits codes. It is obvious that the proposed OSH outperforms both OKH and LSH with a large margin on both datasets. Moreover, we observe that for short code length (16 or 32 bits), the performance of OKH even deteriorates as the number of received chunks increases. In contrast, our proposed OSH achieves stable improvement when more and more data chunks are integrated in the sketch. This is because as more data chunks are received, the sketch we maintained can reflect the characters of the whole training set more completely. Regarding training time comparison, Table 1 shows the accumulated training time of OKH and OSH with 64 bits from the first round to the last round. OSH achieves about three times speed-up on both datasets than OKH, which verifies the efficiency of our method in online setting.

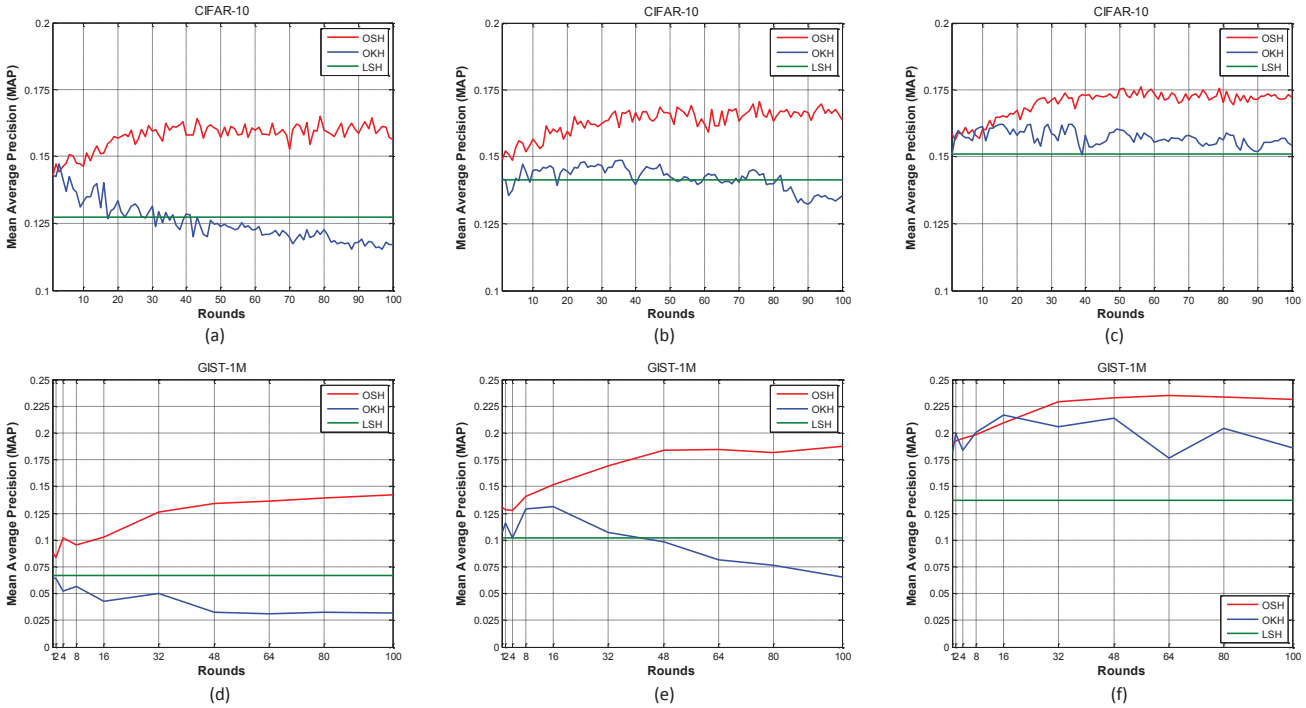

Figure 3. (a)(b)(c) Mean average precision (MAP) on CIFAR-10 dataset at each round with 16, 32, 64 bits. (d)(e)(f) MAP on GIST-1M dataset at each round with 16, 32, 64 bits. (Best viewed in color)

Table 1. Accumulated training time of OKH and OSH (in second)

Dataset	CIFAR-10	GIST-1M
OKH	31.96	490.73
OSH	8.23	180.74

4.4. Evaluation on Scalability

Since one of our initial motivations is to make hash function learning possible when the dataset is too large to be loaded into memory, it is necessary to validate the scalability of the proposed method. To this end, we generate a synthetic dataset consisting of one million samples of 10,000 dimension, denoted as Syn-1M. Assuming 4 byte floating point numbers, the overall space requirement of this dataset is 40GB, which exceeds the memory limits of most PCs. In this experiment we learn hash functions online for this dataset. The dataset is divided into 100 chunks as before. Therefore, we only need to deal with 400M data each round, which can be easily handled by most machines. In the implementation, the sketch size is set to be 100. The accumulated training time for sketching and hash function (64 bits) learning is about 22 minutes (about 12.86 seconds per data chunk). Given the high dimension of the data, this time cost is reasonable and acceptable. The result demonstrates that the proposed OSH is scalable to massive streaming dataset even on a common PC.

5. Conclusion and Future Work

In this paper, we proposed a novel online hash function learning method based on the idea of data sketching. Our approach maintained a small size sketch for a stream of data online and then learned hash functions based on this sketch. Extensive experiments on two real-world datasets and one synthetic dataset demonstrated the superiority of our method in both accuracy and efficiency.

The online sketch used in this work is designed to approximate the covariance matrix, so it is well suited to PCA based hashing. Nevertheless, online sketch for large scale streaming data is a hot topic in computing theory, where many other kinds of sketches in terms of various objective measures have been proposed. For example, online sketches for algorithms like k-means and k-median have been extensively researched in the literature. These sketches may be useful for providing online version for other hashing methods such as [11, 12]. Such extensions can be explored in the future work.

Acknowledgement

This work was supported in part by 863 Program (Grant No. 2014AA015100), National Natural Science Foundation of China (Grant No. 61170127, 61332016, 61370123). The authors would like to thank Dr. Edo Liberty and Dr. Xi Zhang for their constructive suggestion.

References

[1] C. Boutsidis, M. W. Mahoney, and P. Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2009.

[2] M. Charikar. Similarity estimation techniques from rounding algorithm. In *ACM Symposium on Theory of computing*, pages 380–388, 2002.

[3] J. Cheng, C. Leng, J. Wu, H. Cui, and H. Lu. Fast and accurate image matching with cascade hashing for 3d reconstruction. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1–8. IEEE, 2014.

[4] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585, 2006.

[5] P. Drineas and R. Kannan. Pass efficient algorithms for approximating large matrices. In *In Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete algorithms*. SIAM, 2003.

[6] P. Drineas and M. W. Mahoney. On the nystrom method for approximating a gram matrix for improved kernel-based learning. 6:2153–2175, 2005.

[7] A. Frieze, R. Kannan, and S. Vempala. Fast monte-carlo algorithms for finding low-rank approximations. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, 1998.

[8] M. Ghashami and J. M. Phillips. Relative errors for deterministic low-rank matrix approximations. In *SODA*, pages 707–717. SIAM, 2014.

[9] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(12):2916–2929, 2013.

[10] H. Grabner and H. Bischof. On-line boosting and vision. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 260–267. IEEE, 2006.

[11] K. He, F. Wen, and J. Sun. K-means hashing: an affinity-preserving quantization method for learning binary compact codes. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[12] J. Heo, Y. Lee, J. He, S. Chang, and S. Yoon. Spherical hashing. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[13] L.-K. Huang, Q. Yang, and W.-S. Zheng. Online hashing. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1422–1428. AAAI Press, 2013.

[14] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of ACM Symposium on Theory of Computing*, 1998.

[15] A. Jennings, J. McKeown, and A. Jennings. *Matrix computation for engineers and scientists*. J. Wiley, 1992.

[16] A. Joly and O. Buisson. Random maximum margin hashing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[17] W. Kong and W. Li. Isotropic hashing. In *Advances in Neural Information Processing Systems*, 2012.

[18] K. L. Clarkson and D. P. Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*. ACM, 2009.

[19] C. Leng, J. Cheng, T. Yuan, X. Bai, and H. Lu. Learning binary codes with bagging pca. In *Machine Learning and Knowledge Discovery in Databases*, pages 177–192. Springer, 2014.

[20] C. Li, W. Dong, Q. Liu, and X. Zhang. MORES: online incremental multiple-output regression for data streams. *CoRR*, abs/1412.5732, 2014.

[21] E. Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–588. ACM, 2013.

[22] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert. Randomized algorithms for the low-rank approximation of matrices. In *Proceedings of the National Academy of Sciences*, 2007.

[23] J. Lim, D. A. Ross, R.-S. Lin, and M.-H. Yang. Incremental learning for visual tracking. In *Advances in neural information processing systems*, pages 793–800, 2004.

[24] W. Liu, J. Wang, R. Ji, Y. Jiang, and S. Chang. Supervised hashing with kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[25] W. Liu, J. Wang, S. Kumar, and S. Chang. Hashing with graphs. In *Proceedings of the International Conference on Machine Learning*, 2011.

[26] J. Misra and D. Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.

[27] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *Proceedings of the International Conference on Machine Learning*, 2011.

[28] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[29] N. C. Oza. Online bagging and boosting. In *Systems, man and cybernetics, 2005 IEEE international conference on*, volume 3, pages 2340–2345. IEEE, 2005.

[30] R. Polikar, L. Upda, S. S. Upda, and V. Honavar. Learn++: An incremental learning algorithm for supervised neural networks. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 31(4):497–508, 2001.

[31] T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *Proceeding of the Annual IEEE Symposium on Foundations of Computer Science*, 2006.

[32] N. A. Syed, S. Huan, L. Kah, and K. Sung. Incremental learning with support vector machines. 1999.

[33] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[34] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems*, 2008.