Hashing for Distributed Data

Cong Leng CONG.LENG@NLPR.IA.AC.CN
Jiaxiang Wu JIAXIANG.WU@NLPR.IA.AC.CN
Jian Cheng* JCHENG@NLPR.IA.AC.CN
Xi Zhang XI.ZHANG@NLPR.IA.AC.CN
Hanqing Lu LUHQ@NLPR.IA.AC.CN
National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing

Abstract

Recently, hashing based approximate nearest neighbors search has attracted much attention. Extensive centralized hashing algorithms have been proposed and achieved promising performance. However, due to the large scale of many applications, the data is often stored or even collected in a distributed manner. Learning hash functions by aggregating all the data into a fusion center is infeasible because of the prohibitively expensive communication and computation overhead. In this paper, we develop a novel hashing model to learn hash functions in a distributed setting. We cast a centralized hashing model as a set of subproblems with consensus constraints. We find these subproblems can be analytically solved in parallel on the distributed compute nodes. Since no training data is transmitted across the nodes in the learning process, the communication cost of our model is independent to the data size. Extensive experiments on several large scale datasets containing up to 100 million samples demonstrate the efficacy of our method.

1. Introduction

Large scale similarity search is a fundamental building block in many machine learning applications such as clustering, classification and matching (Strecha et al., 2012; Cheng et al., 2014). Traditional nearest neighbor search methods, such as exhaustive search and Kd-tree, become infeasible when the data is huge in size and high in dimensionality. Recently, hashing based approximate nearest neighbor (ANN) search has attracted much attention. Hashing methods encode high dimensional data as compact binary codes so that the distance between two points can be approximated by Hamming distance between their codes. This enables large efficiency gains in both storage and computation speed for similarity search.

Existing hashing approaches can be roughly categorized into data-independent and data-dependent schemes. Early works, such as locality sensitive hashing (LSH) (Indyk & Motwani, 1998; Charikar, 2002) and its variants (Kulis & Grauman, 2009; Datar et al., 2004), construct hash functions based on random projections without considering the input data. More recent works focus on developing data-dependent techniques to learn hash functions by considering the structure of data or side information. Representative approaches include anchor graph hashing (AGH) (Liu et al., 2011), iterative quantization (ITQ) (Gong et al., 2013), spherical hashing (SpH) (Heo et al., 2012), k-means Hashing (KMH) (He et al., 2013) and (Liu et al., 2012; Lin et al., 2013; 2014; Leng et al., 2015).

Most existing hashing algorithms are designed for the centralized setting, or in other words, are single machine approaches. However, because more and more large scale datasets have emerged in real-world applications, the data is often distributed across different locations, such as distributed databases (Corbett et al., 2013), images/videos over the networks, etc. Furthermore, in some applications, the data is inherently distributed. For example, in mobile surveillance (Greenhill & Venkatesh, 2007) and sensor networks (Howard et al., 2002; Liang et al., 2014), the data is collected at distributed sites. In such contexts, in order to get unbiased binary codes for the data, the hash functions should be learned based on the total data. One intuitive way is gathering all data together at a fusion center before training. But it is not a feasible option because of the impractical communication overhead. Besides, directly training on large scale data is often prohibitively expensive in computation, both time and space, which makes it further infeasible for practical applications. As a consequence it has become of central importance to develop hashing algorithms

Proceedings of the 32^{nd} International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37. Copyright 2015 by the author(s).

which are effective in the distritbuted setting. However, to our knowledge, this critical and challenging problem was rarely discussed in the literature.

This paper proposes a novel hashing method for data which is distributed across different nodes whose communication is restricted to the edges of an arbitrary network (*e.g.* Fig.1). Unlike the conventional centralized methods which require gathering the distributed data together to learn common hash functions, our method learns such hash functions in a distributed manner, where each node implements the local computation on its local data, and only exchanges the temporarily learned hash functions with its neighboring nodes. To this end, we cast a centralized hashing model as a set of decentralized subproblems with consensus constraints and ADMM (Hestenes, 1969; Powell, 1967). We show how these subproblems can be efficiently solved with closed-form solutions in parallel. At last, all the nodes obtain consistent hash functions learned from the distributed data. The main contributions of this paper can be summarized as follows:

- We raise a challenging yet rarely discussed problem in hashing, i.e., hashing for distributed data. This issue is essential for the promotion of hashing in real-world large scale retrieval systems.
- We propose a novel approach to learn hash functions in the distributed setting. Since there is no exchange of training data across the nodes in the learning process, the communication cost of our method is small. Additionally, our method can adapt to arbitrary network topology.
- We present extensive experimental evaluations on several large scale image datasets containing up to 100 million samples in a distributed setting. Experimental results verify the proposed method can advance the state-of-the-art both in scale and in accuracy.

2. Preliminary

Vector quantization is a widely used technique in ANN search. A variety of ANN methods, such as iterative quantization (ITQ) (Gong et al., 2013), product quantization (Jegou et al., 2011), Cartesian k-means (Norouzi & Fleet, 2013) can be formulated within a unified framework with minimizing quantization distortion as objective (Ge et al., 2014). In vector quantization, a quantizer maps a vector $\mathbf{x} \in \mathbb{R}^d$ to a codeword $\mathbf{c}$ in a finite codebook. The quantization distortion of a quantizer is defined as:

$$\sum_{\mathbf{x}} \|\mathbf{x} - \mathbf{c}(\mathbf{x})\|^2$$

where $\mathbf{c}(\mathbf{x})$ is the codeword appointed to vector $\mathbf{x}$. If no constraint is imposed on the codebook, minimizing this distortion leads to the classical k-means clustering. As another example, if the codewords are restricted to be taken from the vertexes of a rotating hypercube, minimizing the distortion leads to the well-known hashing method ITQ (Gong et al., 2013).

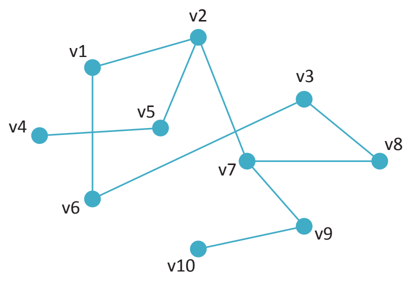

Figure 1. A randomly generated network with 10 nodes. Such a network can be modeled with an undirected and connected graph.

In this work, inspired by (Norouzi & Fleet, 2013), we restrict the codeword to be an additive combination of the columns of a dictionary matrix $C \in \mathbb{R}^{d\times r}$. Formally, let $\mathbf{b}(\mathbf{x})$ be the r-bit binary hashing code of $\mathbf{x}$, then the corresponding assigned codeword is $C\mathbf{b}(\mathbf{x})$. In order to make the optimization tractable, the r columns of dictionary C are constrained to be orthonormal, i.e., $C^TC = I_r$. Taken together with these constraints, minimizing the distortion leads to the objective

$$\begin{aligned} \min_{C,\mathbf{b}} \quad & \sum_{\mathbf{x}} \|\mathbf{x} - C\mathbf{b}(\mathbf{x})\|^2 \\ \text{s.t.} \quad & C^TC = I_r \end{aligned} \tag{1}$$

This problem can be efficiently solved via alternative optimization in a centralized setting. It is worthy noting that the objective (1) is closely related to ITQ (Gong et al., 2013) but with intrinsic differences. In ITQ, the authors divided the learning process into two stages: (a) reduce the dimension of data using PCA to r dimensions, and (b) find an optimal rotation for the data in the projected space to minimize the quantization loss. In contrast, although with orthonormal columns, the matrix C is not squared in (1), which enables us to merge the two stages of ITQ.

3. Distributed Hashing

In this section, we present a distributed hashing model to learn hash functions in a decentralized scenario. In our setting, the data is distributed across a set of P nodes in a network (*e.g.* Fig.1). On the i-th node, there is a local set of n_i data points, denoted in matrix form as $X_i \in \mathbb{R}^{d\times n_i}$, whose columns are data points. The global data $X = \bigcup_{i=1}^P X_i$ is then a concatenation of the local data matrix, i.e. $X = [X_1, X_2, \ldots, X_P]$. Without loss of generality, we assume that the data points are centered to have zero mean. Our goal is to learn binary codes for all the data points. We use $B_i \in \{-1, 1\}^{r\times n_i}$ to denote the code matrix of data points in the i-th node, where r is the code size. $B = [B_1, B_2, \ldots, B_P]$ denotes the global code matrix. tr$(\cdot)$ denotes the trace of a matrix. I_r is an identity matrix of size $r \times r$.

When the data is distributed across the P nodes in an arbitrary network, the objective in (1) can be rewritten as:

$$\begin{aligned} \min_{B_i, C} \quad & \|X - CB\|_F^2 = \sum_{i=1}^{p} \|X_i - CB_i\|_F^2 \\ \text{s.t.} \quad & C^T C = I_r, B_i \in \{-1, 1\}^{n_i \times r} \end{aligned} \tag{2}$$

This can be optimized by alternating between binary coding of the data, i.e., updating B_i's with C fixed, and dictionary update to fit the data, i.e., updating C with B_i's fixed. In the former, fixing C, with simple mathematical derivation, one can show that

$$\|X_i - CB_i\|_F^2 = \|C^T X_i - B_i\|_F^2 + \|{C^{\perp}}^T X_i\|_F^2$$

where $C^{\perp}$ is the orthogonal complement of C. Since the second term $\|{C^{\perp}}^T X_i\|_F^2$ is irrelevant to B_i, it is obvious that $B_i = \text{sgn}(C^T X_i)$ is the optimal solution of the objective as well as the resulting hash functions in this model. Therefore, the code matrices B_i's can be locally updated in parallel on each node if all the nodes have got the dictionary C. In other words, if the dictionary C can also be updated in the distributed setting, then the whole learning process of hashing can be achieved in a completely distributed manner. The difficulty is that, when the codes B_i's are fixed, the dictionary C is shared across all nodes, which makes the problem hard to split.

3.1. Split with Consensus Constraints

In order to make the objective separable, we can replace the global variable C by a set of local variables C_i's, one for each node. By enforcing the consensus constraints $C_i = C_j$ for $i, j \in \{1, 2, \cdots, P\}$, we do not introduce any relaxation into the problem. In particular, with B_i's fixed, the objective in (2) is completely equivalent to:

$$\begin{aligned} \min_{C_i} \quad & \sum_{i=1}^{p} \|X_i - C_i B_i\|_F^2 \\ \text{s.t.} \quad & C_i^T C_i = I_r, \; C_i = C_j, \; \forall i, j \in \{1, 2, \ldots, P\} \end{aligned} \tag{3}$$

The last constraint implies that all the local variables should be consistent. In this way, the additive objective in (2) which does not split is turned into a separable objective. Because of the transitivity between neighboring nodes in a connected graph, we are allowed to consider only the constraints between the neighboring nodes rather than all the constraints. For example, if the consensus constraints between all neighboring nodes are satisfied in Fig.1, then $C_1 = C_4$ is automatically satisfied due to the fact that $C_1 = C_2 = C_5 = C_4$. Based on these observations and to set aside the non-convex orthogonal constraints for the moment, objective (3) can be equivalently reformulated as:

$$\begin{aligned} \min_{C_i} \quad & \sum_{i=1}^{p} \|X_i - C_i B_i\|_F^2 + \mathcal{O}(C_i) \\ \text{s.t.} \quad & C_i = C_{i'}, \quad i' \in \mathcal{N}(i), \quad \forall i \in \{1, 2, \ldots, P\} \end{aligned} \tag{4}$$

where $\mathcal{N}(i)$ represents the neighbors of the i-th node, and $\mathcal{O}$ is defined as an indicator function for whether a matrix has orthonormal columns, that is, for any $Z \in \mathbb{R}^{d \times r}$, if $Z^T Z = I_r$, then $\mathcal{O}(Z) = 0$, otherwise, $\mathcal{O}(Z) = +\infty$.

We now consider the non-convex optimization problem with convex linear constraints. Next we show how the alternating direction method of multipliers (ADMM) (Hestenes, 1969; Powell, 1967) can be efficiently applied to decompose the global problem into several local subproblems and how these subproblems can be solved with closed-form solutions in a distributed manner.

3.2. Distributed Learning

ADMM is a variant of the augmented Lagrangian scheme that blends the decomposability of dual ascent with the superior convergence properties of the method of multipliers (Boyd et al., 2011). For our specific problem, the augmented Lagrangian of (4) is

$$L_\rho(C_i, \Lambda_{i,i'}) = \sum_{i=1}^{p} \left(\|X_i - C_i B_i\|_F^2 + \mathcal{O}(C_i)\right) \tag{5}$$

$$+ \sum_{i=1}^{P} \sum_{i' \in \mathcal{N}(i)} \text{tr}(\Lambda_{i,i'}^T (C_i - C_{i'})) + \frac{\rho}{2} \sum_{i=1}^{P} \sum_{i' \in \mathcal{N}(i)} \|C_i - C_{i'}\|_F^2$$

where $\Lambda_{i,i'}$'s are the Lagrangian multipliers corresponding to the constraints $C_i = C_{i'}$ for $i \in \{1, 2, \ldots, P\}$ and $i' \in \mathcal{N}(i)$. $\rho > 0$ is the penalty parameter of augmented Lagrangian. ADMM solves a problem of this form by repeating the following two steps (Liang et al., 2014):

$$\begin{aligned} C_i^{(k+1)} := \underset{C_i}{\text{argmin}} & \sum_{i=1}^{P} \left(\|X_i - C_i B_i\|_F^2 + \mathcal{O}(C_i)\right) \\ & + \sum_{i=1}^{P} \sum_{i' \in \mathcal{N}(i)} \text{tr}((\Lambda_{i,i'}^{(k)})^T (C_i - C_{i'}^{(k)})) \\ & + \frac{\rho}{2} \sum_{i=1}^{P} \sum_{i' \in \mathcal{N}(i)} \|C_i - C_{i'}^{(k)}\|_F^2 \end{aligned} \tag{6a}$$

$$\begin{aligned} \Lambda_{i,i'}^{(k+1)} := & \Lambda_{i,i'}^{(k)} + \rho(C_i^{(k+1)} - C_{i'}^{(k+1)}) \\ & \forall i \in \{1, \ldots, P\}, i' \in \mathcal{N}(i) \end{aligned} \tag{6b}$$

Despite the algorithm's elegance in form, the subproblems in (6a) can be difficult to solve.

3.2.1. Simplification of Lagrange Multipliers

In the above update rule, assuming each node has t neighboring nodes in average in the network, about Pt Lagrange multipliers have been introduced into the optimization. Such a large number of multipliers enlarge the computation load of algorithm remarkably. However, due to the symmetry of an undirected graph, it is clear that if $i^{'} \in \mathcal{N}(i)$ then $i \in \mathcal{N}(i^{'})$. That is to say, every available constraint in the objective (4) has been considered at least twice, i.e., $C_i = C_{i'}$ and $C_{i'} = C_i$. This implies that we can simplify the Lagrange multipliers. First of all, it is easy to find that the following update rule of multipliers is implicitly contained in (6b):

$$\Lambda_{i^{'},i}^{(k+1)} := \Lambda_{i^{'},i}^{(k)} + \rho(C_{i^{'}}^{(k+1)} - C_i^{(k+1)})$$
$$\forall i^{'} \in \{1,\dots,P\}, i \in \mathcal{N}(i^{'}) \tag{7}$$

For any two adjacent nodes, with (6b) and (7), we have:

$$\Lambda_{i,i^{'}}^{(k+1)} - \Lambda_{i^{'},i}^{(k+1)} = (\Lambda_{i,i^{'}}^{(k)} - \Lambda_{i^{'},i}^{(k)}) + 2\rho(C_i^{(k+1)} - C_{i^{'}}^{(k+1)}) \tag{8}$$

In addition, owing to the symmetric characteristics of the undirected graph, we can rewrite the second term of (5) in another form as:

$$\begin{aligned}
&\sum_{i=1}^{P}\sum_{i^{'}\in\mathcal{N}(i)} \mathrm{tr}(\Lambda_{i,i^{'}}^T(C_i - C_{i^{'}}))\\
=&\sum_{i=1}^{P}\sum_{i^{'}\in\mathcal{N}(i)} \mathrm{tr}(\Lambda_{i,i^{'}}^T C_i) - \sum_{i=1}^{P}\sum_{i^{'}\in\mathcal{N}(i)} \mathrm{tr}(\Lambda_{i,i^{'}}^T C_{i^{'}})\\
=&\sum_{i=1}^{P}\sum_{i^{'}\in\mathcal{N}(i)} \mathrm{tr}(\Lambda_{i,i^{'}}^T C_i) - \sum_{i=1}^{P}\sum_{i^{'}\in\mathcal{N}(i)} \mathrm{tr}(\Lambda_{i^{'},i}^T C_i)\\
=&\sum_{i=1}^{P}\mathrm{tr}(\sum_{i^{'}\in\mathcal{N}(i)} (\Lambda_{i,i^{'}} - \Lambda_{i^{'},i})^T C_i)
\end{aligned} \tag{9}$$

Therefore, by defining P new Lagrange Multipliers Λ_i's as

$$\Lambda_i = \sum_{i^{'}\in\mathcal{N}(i)} (\Lambda_{i,i^{'}} - \Lambda_{i^{'},i}), \forall i \in \{1,2,\cdots,P\},$$

the update of ADMM in (6a) and (6b) can be simplified as:

$$\begin{aligned}
C_i^{(k+1)} := \operatorname*{argmin}_{C_i} &\sum_{i=1}^{P}\left(\|X_i - C_iB_i\|_F^2 + \mathcal{O}(C_i)\right)\\
&+\sum_{i=1}^{P}\mathrm{tr}((\Lambda_i^{(k)})^T C_i) + \frac{\rho}{2}\sum_{i=1}^{P}\sum_{i^{'}\in\mathcal{N}(i)} \|C_i - C_{i^{'}}^{(k)}\|_F^2
\end{aligned} \tag{10a}$$

$$\Lambda_i^{(k+1)} := \Lambda_i^{(k)} + 2\rho\sum_{i^{'}\in\mathcal{N}(i)} (C_i^{(k+1)} - C_{i^{'}}^{(k+1)})$$
$$\forall i \in \{1,\dots,P\}, i^{'} \in \mathcal{N}(i) \tag{10b}$$

In summary, with the derivations in (7)-(9), the original Lagrange multipliers $\Lambda_{i,i^{'}}$ for $i \in \{1,\dots,P\}, i^{'} \in \mathcal{N}(i)$ are reduced to Λ_i for $i \in \{1,2,\cdots,P\}$. The main benefit of this simplification is reducing the computational effort of ADMM. Obviously, the updates of local variables C_i's in (10a) can be separated into P subproblems, and thus can be carried out independently in parallel across the nodes. Although the resulting subproblems are complicated and nonconvex, we find that closed-form optimal solutions can be derived analytically for them.

3.2.2. Solution to Subproblems

Formally, the i-th subproblem can be rewritten in a form of constrained optimization as:

$$\min_{C_i} \|X_i - C_iB_i\|_F^2 + \mathrm{tr}((\Lambda_i^{(k)})^T C_i) + \frac{\rho}{2}\sum_{i^{'}\in\mathcal{N}(i)} \|C_i - C_{i^{'}}^{(k)}\|_F^2$$
$$\text{s.t.} \quad C_i^T C_i = I_r \tag{11}$$

By expanding the quadratic form $\|X_i - C_iB_i\|_F^2$, collecting terms, and neglecting those terms that do not depend on C_i, the subproblem (11) can be rewritten as:

$$\begin{aligned}
\min_{C_i} \quad & -\mathrm{tr}((2B_iX_i^T - (\Lambda_i^{(k)})^T + \rho\sum_{i^{'}\in\mathcal{N}(i)} (C_{i^{'}}^{(k)})^T)C_i)\\
\text{s.t.} \quad & C_i^T C_i = I_r
\end{aligned} \tag{12}$$

For simplicity we define an $r \times d$ matrix S_i as:

$$S_i = 2B_iX_i^T - (\Lambda_i^{(k)})^T + \rho\sum_{i^{'}\in\mathcal{N}(i)} (C_{i^{'}}^{(k)})^T \tag{13}$$

Thus the problem becomes:

$$\max_{C_i} \ \mathrm{tr}(S_iC_i) \qquad \text{s.t.} \ C_i^TC_i = I_r \tag{14}$$

Note that the orthogonal constraint in (14) is a non-linear equality constraint, thus (14) is a non-convex optimization problem. It cannot be solved with the conventional gradient based methods. Besides, the variable C_i is not squared, which makes the optimization more challenging. Nevertheless, we notice problem (14) can be solved analytically with closed-form solution via singular value decomposition (SVD). In specific, we write out the SVD of S_i as:

$$S_i = G_i\Sigma_iH_i^T \tag{15}$$

where $G_i \in \mathbb{R}^{r\times r}$, $H_i \in \mathbb{R}^{d\times r}$ and $\Sigma_i = \mathrm{diag}(\sigma_1,\dots,\sigma_r)$ is a diagonal matrix. The columns of both G_i and H_i are orthonormal, i.e., $G_i^TG_i = I_r$, $H_i^TH_i = I_r$. Since the cyclic permutation leaves the trace unchanged, we have

$$\mathrm{tr}(S_iC_i) = \mathrm{tr}(G_i\Sigma_iH_i^TC_i) = \mathrm{tr}(\Sigma_iH_i^TC_iG_i) \tag{16}$$

Denoting $T_i = H_i^TC_iG_i$, the problem turns into maximizing $\mathrm{tr}(\Sigma_iT_i)$. Before going on, we give the following proposition:

Lemma 1. *For any matrix T of the form $T = H^TCG$ where $H \in \mathbb{R}^{d\times r}, C \in \mathbb{R}^{d\times r}, G \in \mathbb{R}^{r\times r}$ $(d > r)$, if the columns of H, C, R are orthonormal, then any element of the matrix T is not larger than 1.*

Proof. Since G is squared, it can be viewed as an rotation matrix. Denoting $K = CG$, it is easy to find that the columns of K are still orthonormal, i.e., $K^TK = I_r$. For any element of T, we have $T_{m,n} = {H_{(m)}}^T K_{(n)}$, where a subscript (j) means the j-th column of a matrix. According to Cauchy inequality, it is obvious that

$$ {H_{(m)}}^T K_{(n)} \le \|H_{(m)}\| \|K_{(n)}\| = 1, $$

which completes the proof. □

Based on Lemma 1, we have

$$ \mathrm{tr}(\Sigma_i T_i) = \sum_{j=1}^{r} \sigma_j t_{jj} \le \sum_{j=1}^{r} \sigma_j \tag{17} $$

where t_{jj} is the j-th diagonal element of T_i. The trace in (17) is maximized if and only if the diagonal elements of T_i are equal to 1. Hence, there is an obvious optimal solution C_i if it satisfies $T_i = H_i^T C_i G_i = I_r$ and $C_i^T C_i = I_r$. It can be easily seen that

$$ C_i = H_i G_i^T \tag{18} $$

is such a solution.

Algorithm and Convergence:

We describe the whole flowchart of the proposed method which we name *Distributed Hashing* (DisH) in Algorithm 1. It is worth emphasizing that the update of local C_i's (lines 3-5), Λ_i's (line 6) and B_i's (line 8) can be conducted in parallel on each node, which is the key factor for our method being able to work in a distributed setting.

Since a block coordinate descent algorithm is used to optimize the objective function, and the theoretical convergence property of the ADMM update rule is still an open question (Boyd et al., 2011; Bertsekas & Tsitsiklis, 1989), we cannot guarantee that Algorithm 1 reaches the global optimum. Nevertheless, we empirically find that our algorithm is able to fast converge to a reasonable local minima most of the time. To illustrate this, we measure the objective value of (3) at each iteration and see if it decreases or not. We test on two relatively small datasets: CIFAR-10 and GIST-1M, with 32 bits codes. The empirical results are reported in Fig.2. It can be observed that, on both datasets, the objective values decrease as the iterations go on, and converges within less than 10 iterations. These results imply that the convergence of our algorithm is fast in practice. In the experiments section, we will demonstrate that our algorithm has good convergence properties from another point of view.

Algorithm 1 Distributed Hashing

```
Input: Initialize the same dictionaries $C_i$'s, Lagrangian multipliers $\Lambda_i$'s, penalty parameter $\rho$ and number of ADMM iterations $K$ for all nodes. Set the code matrix as $B_i = \mathrm{sgn}(C_i^T X_i)$ for all nodes.
1: repeat
2:   for $k = 1, \dots, K$
3:     Computing $S_i$ with (13) for all nodes;
4:     $[G_i, \Sigma_i, H_i]$ = SVD($S_i$);
5:     Updating $C_i$ with (18) for all nodes;
6:     Updating $\Lambda_i$ with (10b) for all nodes;
7:   end for
8:   Updating $B_i$ with $B_i = \mathrm{sgn}(C_i^T X_i)$ for all nodes;
9: until convergence
```

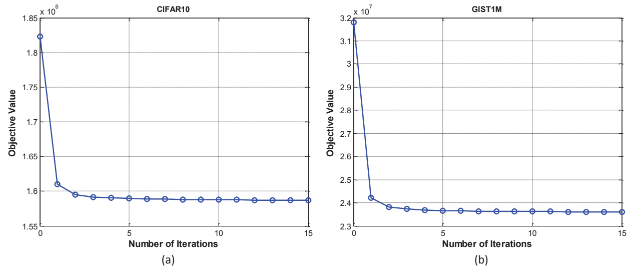

Figure 2. The objective values on (a) CIFAR-10 and (b) GIST-1M decrease as iterations of our algorithm continue. The algorithm converges within 10 iterations on both datasets.

4. Complexity Analysis

Here we analyze the communication and computation complexity of the proposed distributed hashing model. Recall that d denotes the dimension of data, r denotes the length of hashing code, and n_i is the size of data in the i-th node.

• **Communication Complexity:**

In our algorithm, in order to compute S_i in (13) and update Λ_i in (10b), each node needs to share the local dictionary C_i with its neighboring nodes. Supposing the i-th node has t_i neighbors, the communication complexity of this step is $O(t_i dr)$. Interestingly, the local dictionary is the only variable that needs to be communicated during the execution. As a result, the overall communication complexity of each node is $O(t_i dr)$, which is independent to the data size n_i.

• **Computation Complexity:**

For the i-th node, the time complexity to compute S_i in (13) is $O(n_i dr)$. The following SVD for S_i requires $O(dr^2)$ time. The complexity of updating the local variable C_i with (18) is $O(dr^2)$. Thus, the overall time complexity of updating the dictionary is $O(Kn_i dr + Kdr^2)$, where K is the number of iterations in ADMM. The update of hashing code on each node takes up $O(n_i dr)$ in time. To summarize, the time complexity of computation on each node is linear in both size n_i and dimension d of the data.

Table 1. Comparisons of mean average precision (MAP) with different code sizes on CIFAR-10 and GIST-1M. The compared state-of-the-arts are all centralized methods. DisH[1] denotes distributed hashing with the first search strategy, while DisH[2] denotes that with the second search strategy. The standard deviations of DisH[2] are reported in the last line.

Methods	CIFAR-10					GIST-1M				
	8-bits	16-bits	24-bits	32-bits	64-bits	8-bits	16-bits	24-bits	32-bits	64-bits
AGH	**0.1661**	0.1654	0.1613	0.1585	0.1490	0.0807	0.1224	0.1461	0.1572	0.1805
KMH	0.1493	0.1577	0.1632	0.1636	0.1624	**0.1015**	0.1362	0.1497	0.1560	0.1565
SpH	0.1306	0.1435	0.1507	0.1528	0.1632	0.0832	0.1213	0.1489	0.1544	0.2137
ITQ	0.1534	**0.1690**	**0.1710**	**0.1734**	**0.1781**	0.1008	**0.1406**	**0.1704**	**0.1878**	**0.2188**
DisH[1]	**0.1543**	**0.1649**	**0.1698**	**0.1724**	**0.1780**	**0.1013**	**0.1401**	**0.1656**	**0.1842**	**0.2118**
DisH[2]	**0.1557** $\pm$ .0003	**0.1636** $\pm$ .0003	**0.1700** $\pm$ .0001	**0.1708** $\pm$ .0003	**0.1778** $\pm$ .0002	**0.1020** $\pm$ .0002	**0.1478** $\pm$ .0004	**0.1690** $\pm$ .0002	**0.1834** $\pm$ .0005	**0.2166** $\pm$ .0006

5. Experiments

In order to evaluate the performance of the proposed distributed hashing (DisH), we conduct a series of experiments on different datasets for retrieval. In all experiments, we assume the data is distributed across different nodes in a network. In specific, we randomly construct a network with 10 nodes, as shown in Fig.1. Each node has a 2.50GHz Intel Xeon CPU. The implementation of our distributed system is based on the Distributed Computing Engine of MATLAB in Linux.

5.1. Experimental Setting

Ideally, the resulting local dictionaries C_i's among the nodes will be consistent because they are learned with the consensus constraints. However, since the solution obtained by ADMM is not theoretically guaranteed to be the global optimum (Boyd et al., 2011), the consistency of the local dictionaries needs to be verified. To this end, we test two search strategies for the proposed distributed hashing. In the first strategy, when a query $\mathbf{q}$ arrives, it will be broadcasted to all the nodes. The query will be embedded to binary code with the corresponding local dictionary on each node, i.e., $\mathbf{b}_i = \mathrm{sgn}(C_i^T \mathbf{q})$. Subsequently, the Hamming distances between $\mathbf{b}_i$ and samples on each node will be calculated. In the second strategy, we assign one of the local dictionaries as the final dictionary C, then both the queries and database will be hashed with this common dictionary. In the following comparisons, we denote the first search strategy of our method as DisH^1, while the second strategy as DisH^2. The experiments with DisH^2 are conducted 10 times, where the i-th dictionary C_i is assigned as the final dictionary in the i-th execution. We report the average results and their standard deviations of DisH^2. Obviously, if the local dictionaries are consistent, the standard deviations of DisH^2 will be very small and the results derived by DisH^1 and DisH^2 will be very close.

To perform fair evaluation, we adopt the Hamming ranking strategy which is commonly used in the literature (Wang et al., 2012; Liu et al., 2011). All points in the database are ranked according to their Hamming distance to the query and the top ranked samples will be returned. Throughout the experiments, we empirically set the penalty parameter ρ as $\rho = 100$ and the number of ADMM iterations K as $K = 5$.

5.2. Compare with Centralized Methods

In this simple experiment, we aim to verify that hashing in a distributed setting won't loss much quality compared with that in a centralized setting. We compare our distributed method against state-of-the-art centralized hashing methods including AGH (Liu et al., 2011), KMH (He et al., 2013), SpH (Heo et al., 2012) and ITQ (Gong et al., 2013).

We choose two relatively small benchmarks: CIFAR-10[1] and GIST-1M[2] for this experiment. CIFAR-10 consists of 60K 32×32 color images in 10 classes, with 6,000 images per class. We extract 512 dimensional GIST descriptor (Oliva & Torralba, 2001) 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 is assigned a class label, the ground truth is defined as semantic neighbors based on label agreement. For GIST-1M dataset, top 2 percentile nearest neighbors in Euclidean space are taken as ground truth. The data is evenly divided into 10 splits and distributed across the nodes. For all compared centralized methods, because the datasets are small, it allows us to assemble all the training data at a single node and learn hash functions on it.

Table 1 shows the Hamming ranking performance measured by Mean Average Precision (MAP) with different code sizes on two datasets. From these results we can find that, on both datasets, our method outperforms most of

[1]`http://www.cs.toronto.edu/~kriz/`
[2]`http://corpus-texmex.irisa.fr/`

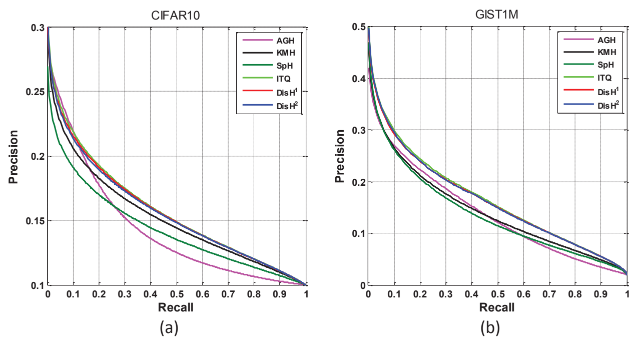

Figure 3. The precision-recall curves with 32 bits on two datasets (a) CIFAR-10 and (b) GIST-1M. Best viewed in color.

the centralized baselines. The two versions of our method DisH^1 and DisH^2 achieve nearly the same accuracy. Besides, the standard deviations of DisH^2 are typically negligible (less than 0.001 in all cases). This implies the consistency of the learned local dictionaries, which also implies the ADMM converges well in our algorithm. It can be observed that the performance of our algorithm is very close to that of ITQ which in most cases is the best competitor in the centralized methods. As we have explained above, the hashing model represented by objective (1) is closely related to ITQ, and our method essentially optimizes (1) in a distributed manner. The comparisons with ITQ suggest that learning hash functions in a distributed setting does not loss much quality compared to the centralized setting. Moreover, our method and ITQ outperform other state-of-the-art baselines for most code sizes, which indicates that minimizing quantization distortion is effective for binary codes learning.

Fig.3 shows the complete precision-recall curves on two datasets with 32 bits code. These detailed results are consistent with the trends discovered in Table 1. The curves of our methods and ITQ almost overlap, and outperform other competitors with a large margin.

5.3. Evaluation on Large Scale Datasets

In the second experiment, we measure the accuracy of our method on two large scale datasets: MNIST-8M[3] and SIFT-10M[4]. MINST-8M consists of 8 million 784-dimensional samples associated with handwritten digits from ‘0’ to ‘9’. SIFT-10M consists of 10 million local SIFT descriptors extracted from random images. We sample 2,000 data points to be query set and employ the rest as database and training set. For the MNIST-8M dataset, the ground truth is defined as semantic neighbors. For SIFT-10M, top 2‰ nearest neighbors in Euclidean space are taken as ground truth. The data is evenly distributed across 10 nodes.

[3]`http://www.csie.ntu.edu.tw/~cjlin/`

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

To our knowledge, this work could be the first attempt to learn hash functions in a distributed setting, therefore we cannot find closely related work to compare with in this setting. It is also infeasible to transmit the data across the nodes for such large datasets. In fact, the only existing hash method which can work in such a distributed setting may be the data-independent LSH (Charikar, 2002), in which the hash functions are not learned but randomly selected from a locality-sensitive function family. Thus, we compare the performance of our method with LSH in this experiment. Since computing MAP is very slow on large scale datasets, we report the mean precision of top 50-1000 ranked neighbors in Hamming space.

Fig.4 shows the precision on MNIST-8M. Our method attains high accuracy, and yields to better accuracy as the code size increases. As shown in Fig.4(d), the precision of top-50 reaches 95.4% with 64 bits. The improvement over LSH is significant. For example, the gain in precision of top-1000 of our method ranges from 16% to 87% over LSH with different code sizes. This is because the learned hash functions can capture more information about the data structure than the randomly generated hash functions in LSH. Note that both the performance of DisH^1 (black line) and DisH^2 (red line) are reported, but the two lines almost completely overlap on this dataset. The standard derivations of DisH^2 are also reported as error bars on the red line, which are also negligible as in Table 1. These phenomenons again verify the consistency of the local dictionaries. Fig.5 shows the precision on SIFT-10M. Similar trends can be found as in Fig.4. Again, it can be clearly observed that both DisH^1 and DisH^2 outperform LSH with a large margin. For example, as shown in Fig.5(b), our method accomplishes about 2 times higher precision than LSH with 24 bits code. All these results indicate that the proposed method is very effective to learn hash functions in a distributed setting, and it offers another better choice for real-world distributed retrieval systems beyond LSH.

5.4. Evaluation of Efficiency

In order to evaluate the time efficiency of our method, we conduct experiments on a larger dataset SIFT-100M[5], which consists of 100 million SIFT descriptors extracted from random images. We test on multiple subsets of different sizes extracted from SIFT-100M, to verify whether the time consumption grows linear with respect to the number of data points. We also vary the number of nodes to see the how training time can be shorten with more nodes being used. When the same number of nodes are employed, the training time might be related to the topology of the network. For convenience, we restrict the network to be a star topology in this experiment.

[5]`http://corpus-texmex.irisa.fr/`

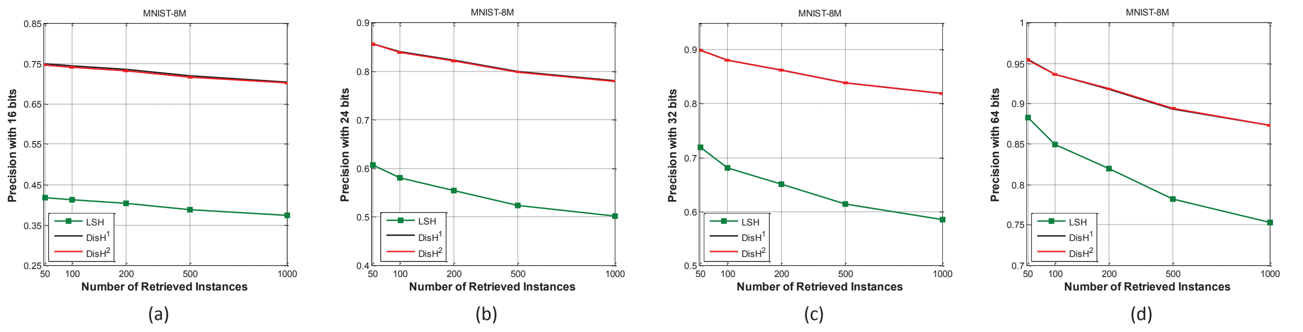

Figure 4. Results on **MNIST-8M**. Hamming ranking precision of top 50, 100, 200, 500 and 1000 ranked samples with different code sizes: (a) 16 bits, (b) 24 bits, (c) 32 bits and (d) 64 bits. The black line of DisH^1 and the red line of DisH^2 overlap almost everywhere.

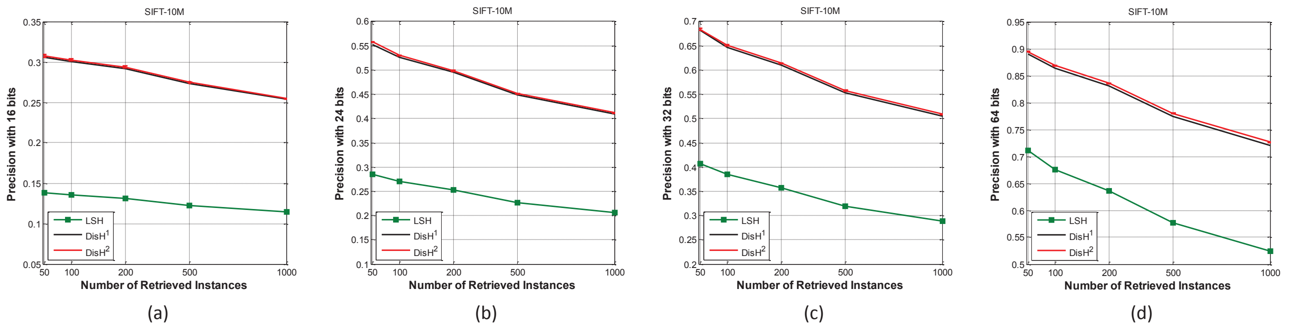

Figure 5. Results on **SIFT-10M**. Hamming ranking precision of top 50, 100, 200, 500 and 1000 ranked samples with different code sizes: (a) 16 bits, (b) 24 bits, (c) 32 bits and (d) 64 bits. Best viewed in color.

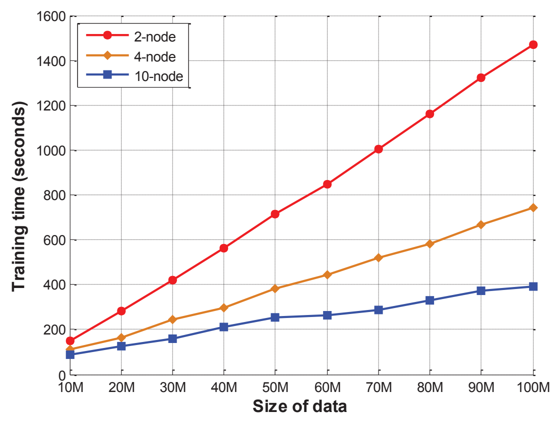

Figure 6. Training time with different data sizes when different number of side nodes involved.

Fig.6 presents the training time under different settings. Roughly speaking, the larger the data size, the higher the running time. Meanwhile, the more nodes involved to computation, the shorter the running time. This phenomenon is natural and easy to understand, which also agrees with our previous analysis on computation complexity in Section 4. As we have analyzed, the time complexity of computation on each node is linear to the number of data on it. In Fig.6, we can find that the training time increases linearly as the data size increases. With more nodes involved to the computation, the size of data distributed to each node will be smaller, thus the corresponding training time will be less. It is worth noting that the training time on the dataset of 100 million samples is about 391 seconds when 10 nodes are employed. This time cost is very small and really acceptable given the large scale of the dataset. These results demonstrate that the proposed distributed hashing is scalable to massive data in real-world applications.

6. Conclusion

In this paper, we proposed a distributed hashing model to learn binary codes for distributed data. We casted a centralized hashing model into a set of decentralized subproblems with consensus constraints and showed how these subproblems can be solved in parallel in a distributed manner. Our method could adapt to arbitrary network topologies with small communication and computation cost. Extensive experiments on several large scale datasets fully verified the efficacy of the proposed method.

Acknowledgement

This work was supported in part by 863 Program (Grant No. 2014AA015100), National Natural Science Foundation of China (Grant No. 61170127, 61332016). Jian Cheng is the corresponding author.

References

Bertsekas, Dimitri P and Tsitsiklis, John N. *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ, 1989.

Boyd, Stephen, Parikh, Neal, Chu, Eric, Peleato, Borja, and Eckstein, Jonathan. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

Charikar, M. Similarity estimation techniques from rounding algorithm. In *ACM symposium on Theory of computing*, pp. 380–388, 2002.

Cheng, Jian, Leng, Cong, Wu, Jiaxiang, Cui, Hainan, and Lu, Hanqing. Fast and accurate image matching with cascade hashing for 3d reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

Corbett, James C, Dean, Jeffrey, Epstein, Michael, Fikes, Andrew, Frost, Christopher, Furman, JJ, Ghemawat, Sanjay, Gubarev, Andrey, Heiser, Christopher, Hochschild, Peter, et al. Spanner: Googles globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V.S. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*, pp. 253–262, 2004.

Ge, Tiezheng, He, Kaiming, Ke, Qifa, and Sun, Jian. Optimized product quantization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(4):744–755, April 2014.

Gong, Yunchao, Lazebnik, Svetlana, Gordo, Albert, and Perronnin, Florent. 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.

Greenhill, Stewart and Venkatesh, Svetha. Distributed query processing for mobile surveillance. In *Proceedings of the 15th international conference on Multimedia*, pp. 413–422. ACM, 2007.

He, Kaiming, Wen, Fang, and Sun, Jian. K-means hashing: an affinity-preserving quantization method for learning binary compact codes. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

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

Hestenes, Magnus R. Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5):303–320, 1969.

Howard, Andrew, Matarić, Maja J, and Sukhatme, Gaurav S. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Distributed autonomous robotic systems 5*, pp. 299–308. Springer, 2002.

Indyk, P. and Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, 1998.

Jegou, Herve, Douze, Matthijs, and Schmid, Cordelia. Product quantization for nearest neighbor search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(1):117–128, 2011.

Kulis, B. and Grauman, K. Kernelized locality-sensitive hashing for scalable image search. In *International Conference on Computer Vision*, 2009.

Leng, Cong, Wu, Jiaxiang, Cheng, Jian, Bai, Xiao, and Lu, Hanqing. Online sketching hashing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

Liang, Junli, Zhang, Miaohua, Zeng, Xianyu, and Yu, Guoyang. Distributed dictionary learning for sparse representation in sensor networks. *Image Processing, IEEE Transactions on*, 23(6): 2528 – 2541, 2014.

Lin, Guosheng, Shen, Chunhua, Suter, David, and Hengel, Anton van den. A general two-step approach to learning-based hashing. In *IEEE International Conference on Computer Vision (International Conference on Computer Vision)*, 2013.

Lin, Guosheng, Shen, Chunhua, Shi, Qinfeng, van den Hengel, Anton, and Suter, David. Fast supervised hashing with decision trees for high-dimensional data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1971–1978, 2014.

Liu, W., Wang, J., Kumar, S., and Chang, S. Hashing with graphs. In *International Conference on Machine Learning*, 2011.

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

Norouzi, M. and Fleet, D.J. Cartesian k-means. In *CVPR*, pp. 3017–3024, 2013.

Oliva, Aude and Torralba, Antonio. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42(3):145–175, 2001.

Powell, Michael JD. *A method for non-linear constraints in minimization problems*. UKAEA, 1967.

Strecha, Christoph, Bronstein, Alexander M, Bronstein, Michael M, and Fua, Pascal. Ldahash: Improved matching with smaller descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):66–78, 2012.

Wang, J., Kumar, S., and Chang, S. Semi-supervised hashing for large-scale search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(12):2393–2406, 2012.