2013 IEEE International Conference on Computer Vision

Learning Hash Codes with Listwise Supervision

Jun Wang
Business Analytics and Mathematical Sciences
IBM T. J. Watson Research Center
wangjun@us.ibm.com

Wei Liu
Multimedia Analytics
IBM T. J. Watson Research Center
weiliu@us.ibm.com

Andy X. Sun
School of Industrial and Systems Engineering
Georgia Institute of Technology
andy.sun@isye.gatech.edu

Yu-Gang Jiang
School of Computer Science
Fudan University
ygj@fudan.edu.cn

Abstract

Hashing techniques have been intensively investigated in the design of highly efficient search engines for large-scale computer vision applications. Compared with prior approximate nearest neighbor search approaches like tree-based indexing, hashing-based search schemes have prominent advantages in terms of both storage and computational efficiencies. Moreover, the procedure of devising hash functions can be easily incorporated into sophisticated machine learning tools, leading to data-dependent and task-specific compact hash codes. Therefore, a number of learning paradigms, ranging from unsupervised to supervised, have been applied to compose appropriate hash functions. However, most of the existing hash function learning methods either treat hash function design as a classification problem or generate binary codes to satisfy pairwise supervision, and have not yet directly optimized the search accuracy. In this paper, we propose to leverage listwise supervision into a principled hash function learning framework. In particular, the ranking information is represented by a set of rank triplets that can be used to assess the quality of ranking. Simple linear projection-based hash functions are solved efficiently through maximizing the ranking quality over the training data. We carry out experiments on large image datasets with size up to one million and compare with the state-of-the-art hashing techniques. The extensive results corroborate that our learned hash codes via listwise supervision can provide superior search accuracy without incurring heavy computational overhead.

1. Introduction

Due to the rapidly growing scale and dimensionality of gigantic data, such as images and videos, retrieving relevant samples from large-scale data collections has become an inevitable need in many practical applications. Instead of exhaustively searching the most similar samples to a query, approximate nearest neighbor (ANN) search methods such as hashing-based techniques have been studied extensively, particularly in the domain of similar image search [10, 21].

Briefly speaking, the objective of hashing is to map an original D-dimensional data space $\mathbb{R}^D$ to a binary *Hamming space* $\mathbb{B}^K$, where each data point is represented by a binary hash code (i.e., a K-bit hash key) and the entire data set is mapped to a table with hash keys as entries, namely a hash table. Most of the early hashing techniques including locality sensitive hashing [5] and MinHash are data-independent random approaches, which do not perform well in applications like image retrieval and search [21].

Recently, semi-supervsied/supervised learning algorithms have been employed to design more effective hash functions and many new hashing methods have been proposed [6, 9, 12, 14, 17, 18, 21, 22]. These *learning to hash* methods are either *pointwise* or *pairwise*, which—in other words—leverage instance-level labels or pairwise relations between instances into the learning procedure. Their hashing objectives are to preserve the pointwise or pairwise label information in the learned Hamming space. Although promising performance has been shown from these methods, we argue that their objectives are sub-optimal in search tasks since ranking information was not fully utilized.

In this paper, we propose a novel framework for learning hash functions that preserve ground-truth orders of ranking lists. Our learning procedure takes into account the ranking orders, not just instance-level or pairwise label information that was widely used in the prior works. The proposed framework—namely *ranking-based supervised hashing* (RSH)—has three key steps, as demonstrated by the conceptual diagram in Figure 1. First, the given ground-

1550-5499/13 $31.00 © 2013 IEEE
DOI 10.1109/ICCV.2013.377

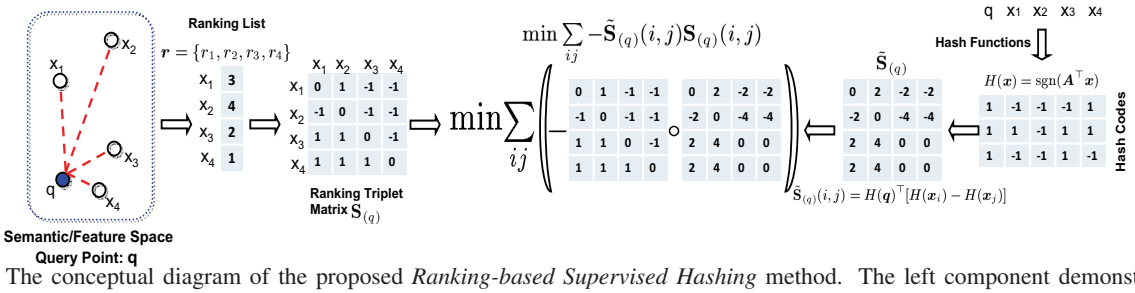


Figure 1. The conceptual diagram of the proposed *Ranking-based Supervised Hashing* method. The left component demonstrates the procedure of deriving ground-truth ranking list $\mathbf{r}$ using the semantic relevance or feature similarity/distance, and then converting it to a *triplet matrix* $\mathbf{S}_{(q)}$ for a given query q. The right component describes the estimation of a relaxed ranking *triplet matrix* $\tilde{\mathbf{S}}_{(q)}$ from the binary hash codes. The central component shows the objective of minimizing the inconsistency between the two ranking *triplet matrices*.

truth ranking lists for individual queries are converted to a *triplet* representation where each triplet $\mathbf{S}(\mathbf{q}; \mathbf{x}_i, \mathbf{x}_j)$ indicates the order of each instance pair $(\mathbf{x}_i, \mathbf{x}_j)$ given a certain query $\mathbf{q}$. Second, given the hash code $H(\mathbf{q})$ for a query $\mathbf{q}$ and $H(\mathbf{x}_i), H(\mathbf{x}_j)$ for an instance pair $\mathbf{x}_i, \mathbf{x}_j$, we apply inner products to compute the similarities among hash codes and then derive the rank triplet $\tilde{\mathbf{S}}\left(H(\mathbf{q}); H(\mathbf{x}_i), H(\mathbf{x}_j)\right)$ in the Hamming space. The final step is to minimize the inconsistency between the ground-truth rank triplets and the ones derived from the corresponding hash codes, which implicitly preserves the ground-truth ranking orders in the Hamming space. After relaxation of the non-differentiable loss function, we show that the optimal solution can be efficiently solved using the *Augmented Lagrange Multipliers* (ALM) method with a convergence guarantee. Empirical studies are performed over three datasets, and results clearly show that the proposed RSH method can generate higher-quality search results.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of related works in hash function design. Section 3 describes the conversion from ranking lists to the triplet representation, and then defines a loss function measuring the inconsistency between the ground-truth ranking list and the ranking list derived in the Hamming space. Section 4 presents the formulation as well as our solution for RSH. The experiments are reported in Section 5 followed by conclusions in Section 6.

2. Related Works

Tremendous efforts have been paid to design more effective hashing techniques in the past years. This section gives a brief introduction of existing hashing techniques. Specifically, we focus on two categories of methods, which are more related to this paper: *label-dependent hashing* and *ranking order statistics based hashing*.

2.1. Label-Dependent Hashing

Realizing that semantic relevance or similarity among data can not be fully defined by a single distance metric, several researchers explored supervision information to design task-specific hash functions. As briefly mentioned earlier, there are two types of semi-supervised or supervised hashing methods. The first category can be named as *pointwise* approaches since each hash function is essentially treated as a binary classifier and the provided label information is exploited to guide the hash function design. For instance, boosted similarity sensitive coding (BSSC) [18] is one of the earliest efforts to incorporate label information, which attempts to learn a weighted Hamming embedding for a specific search task.

The second type can be categorized as *pairwise* approaches since the learning algorithms usually take the pairwise supervision information. For instance, in [21], the authors proposed a semi-supervise hashing (SSH) technique, which aims to maximize the empirical fitness over the labeled sample pairs, while also maximize the information gain from each hash bit to avoid overfitting. Thereby, SSH tends to derive hashing functions with the highest accuracy over training data and a balanced partitioning over the entire dataset [21]. Besides SSH, many other recent methods also belong to this category, such as binary reconstructive embedding [9], minimal loss hashing [14], complementary hashing [26], and distance metric learning based hashing [10].

In summary, the existing supervised and semi-supervised hashing methods often use the pairwise label information to pursue the optimal hash functions. The general objective is to encode similar point pairs to the same hash bucket while maximizing the difference of hash codes of dissimilar point pairs. One prominent advantage of such pairwise approaches is that the hashing formulation is relatively straightforward, and therefore many existing theories and algorithms can be easily migrated to this domain. However, these approaches do not directly optimize on ranking lists, which is critical to assess the search quality in practical applications.

2.2. Order Preserving Hashing

Compared with the pairwise approaches, there are very few works using ranking order information. The concomi-

tant min-hashing (CMH) method was proposed as a natural extension of the conventional MinHash family [4]. CMH exploits the theory of concomitant order statistics to develop locality sensitive hash functions, which can improve the performance under the Cosine similarity measure. More recently, winner-takes-all hashing (WTAH) explores partial order statistics and encodes relative orders of feature dimensions to develop randomized data-dependent hash functions [27]. WTAH produces a sparse embedding and the resulting Hamming distance closely correlates with the ranking similarity measure. However, both CMH and WTAH belong to the randomized hashing family since they use either random projections or random permutations. In other words, they are not really supervised methods explicitly using ranking information. Regarding the importance of preserving ranking orders, a tree- based method [16] was proposed. More recently, Norouzi et al. developed a hamming metric learning framework to minimize the piecewise-smooth upper bound on a triplet ranking loss [15].

3. Listwise Supervision using Triplets

In this section, we first define notations that will be used throughout this paper. Then we propose a *triplet* representation to formulate our objective of preserving ranking orders.

3.1. Listwise Supervision

Assume the dataset $\mathcal{X} = \{\mathbf{x}_n\}_{n=1}^N$ has N points and each point is in a D-dimensional real space, i.e., $\mathbf{x}_n \in \mathbb{R}^D$. In addition, we have a query set $\mathcal{Q} = \{\mathbf{q}_m\}_{m=1}^M$. For any specific query point $\mathbf{q}_m$, we can derive a ranking list over $\mathcal{X}$, which can be written as a vector as

$$r(\mathbf{q}_m, \mathcal{X}) = (r_1^m, \cdots, r_n^m, \cdots, r_N^m), \tag{1}$$

where each element r_n^m falls into the integer range $[1, N]$ and no two elements share the same value. If $r_i^m < r_j^m$ $(i, j = 1, \cdots, N)$, it indicates sample $\mathbf{x}_i$ has higher rank than $\mathbf{x}_j$, which means $\mathbf{x}_i$ is more relevant or similar to $\mathbf{q}_m$ than $\mathbf{x}_j$. If given a similarity function $sim(\cdot)$, we can easily derive the ranking list $(r_1^m, \cdots, r_n^m)$ associated with a query sample $\mathbf{q}_m$, where the order is estimated as $r_i^m < r_j^m$ if $sim(\mathbf{x}_i, \mathbf{q}_m) > sim(\mathbf{x}_j, \mathbf{q}_m)$. Similarly, if $sim(\mathbf{x}_i, \mathbf{q}_m) < sim(\mathbf{x}_j, \mathbf{q}_m)$, we can derive $r_i^m > r_j^m$.

Though there could be multiple database points with the same similar measure to the query point, in this paper, we ignore the equal case $k(\mathbf{x}_i, \mathbf{q}_m) = k(\mathbf{x}_i, \mathbf{q}_m)$ and assume each database point has an exact ranking order. The ground-truth ranking list in Eq. (1) can be easily derived if a similarity measure between datapoints is predefined. However, if given the semantic label information, it is also fairly straightforward to convert semantic labels to ranking lists through counting the commonly shared labels between the query point and the database points.

In our formulation for learning hash functions, we will leverage the above supervision information in the form of a ranking list to design ranking-preserving binary hash codes.

3.2. Conversion from a Ranking List to Triplets

Compared to those pairwise based hash learning approaches, using the listwise supervision has a clear advantage since optimizing the ranking list can directly improve the quality of nearest neighbor search. However, the main challenge is how to effectively leverage such ranking information into the learning framework. In the previous research of *learning to rank* [11], the framework using the ranked list to train a ranking function, namely listwise approaches, has been well studied. Typically, the objective is to minimize an expected loss function, which intrinsically measure the difference between the ground-truth ranking and the ranking derived from the rank function [25]. One of the fundamental difference in learning hash function lies in that the hash function only generates binary codes, instead of explicit ranking scores.

Here we present an efficient way to translate the ground-truth ranking list into a set of *rank triplets*, which can be easily fed into the hash function learning paradigm. Recall we have the ground-truth ranking list for a query $\mathbf{q}_m$ represented as a vector $(r_1^m, \cdots, r_n^m, \cdots, r_N^m)$. Therefore, we use a rank triplet $S(\mathbf{q}_m; \mathbf{x}_i, \mathbf{x}_j) \in \mathbb{R}$ to represent the listwise supervision and the value of $S(\mathbf{q}_m; \mathbf{x}_i, \mathbf{x}_j)$ is defined as

$$S(\mathbf{q}_m; \mathbf{x}_i, \mathbf{x}_j) = \begin{cases} 1 & : \quad r_i^q < r_j^q \\ -1 & : \quad r_i^q > r_j^q \\ 0 & : \quad r_i^q = r_j^q. \end{cases} \tag{2}$$

It is straightforward to see the triplet $S(\mathbf{q}_m; \mathbf{x}_i, \mathbf{x}_j)$ associates with the rank order for sample $\mathbf{x}_i, \mathbf{x}_j$ given a query point $\mathbf{q}_m$. Hence, the ranking list $r(\mathbf{q}_m, \mathcal{X})$ can be converted to a set of triplets, which can be represented in a matrix form as $\mathbf{S}_{(\mathbf{q}_m)} \in \mathbb{R}^{N \times N}$, namely a *triplet matrix*, where each element satisfies

$$S_{(\mathbf{q}_m)}(i, j) = S(\mathbf{q}_m; \mathbf{x}_i, \mathbf{x}_j), i, j = 1, \cdots, N.$$

Since each triplet matrix $\mathbf{S}_{(\mathbf{q}_m)}$ has one-to-one correspondence to the ranking list $(r_1^m, \cdots, r_n^m, \cdots, r_N^m)$. with respect to a query point $\mathbf{q}_m$, Hence for a set of query points $\mathcal{Q} = \{\mathbf{q}_m\}_{m=1}^M$, we can derive a *triplet tensor*, i.e., a set of triplet matrices $\mathbf{S} = \{\mathbf{S}_{(\mathbf{q}_m)}\} \in \mathbb{R}^{M \times N \times N}$. In particular, the element of the triplet tensor is defined as $\mathbf{S}_{mij} = \mathbf{S}_{(\mathbf{q}_m)}(i, j) = S(\mathbf{q}_m; \mathbf{x}_i, \mathbf{x}_j)$.

3.3. Loss Function by Triplets Representation

Like most of the listwise approaches, a loss function is defined to measure the difference between two ranking lists. Given a query point $\mathbf{q}_m$ and the data $\mathcal{X}$, assume a function

$f(\cdot) : \mathcal{X} \to \mathcal{R}$ can be used to generate a permutation, i.e. a ranking list, over $\mathcal{X}$ as:

$$\begin{aligned}\tilde{r}(\mathbf{q}_m, \mathcal{X}) &= f(\mathbf{q}_m, \mathcal{X}) \\ &= (f(\mathbf{q}_m, \mathbf{x}_1), \cdots, f(\mathbf{q}_m, \mathbf{x}_n), \cdots, f(\mathbf{q}_m, \mathbf{x}_N))\end{aligned} \tag{3}$$

Let $P(\mathbf{x}, \mathbf{r})$ be the unknown joint distribution of $\mathbf{x} \in \mathcal{X}$ and $\mathbf{r} \in \mathcal{R}$ and $V(f(\mathbf{x}), \mathbf{r})$ is a loss function. Then the total loss for the function $f(\cdot)$ can be computed as

$$L(f, \mathcal{X}, \mathcal{Y}) = \int_{\mathcal{Y}} \int_{\mathcal{X} \times \mathcal{R}} V(f(\mathbf{x}), \mathbf{r}) dP(\mathbf{x}, \mathbf{r}) \tag{4}$$

Since the ranking problem can be treated as a special case of classification or regression, the placement of each sample is the target for prediction [19]. The above loss function essentially measures average precision in a cost-insensitive manner, where the loss is equally computed for all wrongly ranked samples without considering the positions in the ranking list. This formulation of loss function has been studied in the literature of *learning to rank* [11]. Note that the ranking function $f(.)$ typically first assigns a prediction score to each sample, and then perform sorting process over such scores to derive the final permutation. However, due to the natural of sorting procedure, minimizing such a loss function is intractable [25].

To tackle this challenging issue, here we propose to use the triplets to assess the quality of a ranking list without sorting the samples. First define $g(\cdot)$ as a scoring function, which measures the similarity between two samples, e.g., $g(\mathbf{x}_i, \mathbf{q}_m)$. If a sample $\mathbf{x}_i$ is ranked ahead of $\mathbf{x}_j$, indicated by a triplet as $S_{(\mathbf{q}_m)}(i, j) = 1$, it is expected that the similarity function satisfies $g(\mathbf{x}_i, \mathbf{q}_m) > g(\mathbf{x}_j, \mathbf{q}_m)$, otherwise $g(\mathbf{x}_i, \mathbf{q}_m) < g(\mathbf{x}_j, \mathbf{q}_m)$. Hence, we can have the following loss function measuring the quality of the ranking list derived from the scoring function $g(\cdot)$

$$L(g, \mathcal{X}, \mathcal{Y}) = -\sum_{m,i,j} \tilde{S}_{(\mathbf{q}_m)}(i, j) S_{(\mathbf{q}_m)}(i, j) \tag{5}$$

where $\tilde{S}_{(\mathbf{q}_m)}(i, j)$ is a ranking triplet computed from the similarity scores $g(\mathbf{x}_i, \mathbf{q}_m), g(\mathbf{x}_j, \mathbf{q}_m)$ and $S_{(\mathbf{q}_m)}(i, j)$ is the ground-truth. Note that if the triplets $\tilde{S}_{(\mathbf{q}_m)}(i, j), S_{(\mathbf{q}_m)}(i, j)$ agree with each other, they contribute negative loss, and contribute positive loss otherwise. Intuitively, minimizing the above loss function will lead to maximum consistency between ground-truth triplets and the triplets derived from $g(\cdot)$. Recall the definition of the ranking triplet in Eq.(2), we can easily calculate the triplet value using the similarity measure as

$$\tilde{S}_{(\mathbf{q}_m)}(i, j) = \operatorname{sgn}\left(g(\mathbf{x}_j, \mathbf{q}_m) - g(\mathbf{x}_i, \mathbf{q}_m)\right). \tag{6}$$

However, directly using the above triplet measure will make the loss function non-differentiable and hard to optimize. A typical way is to relax $\tilde{S}$ and use the *signed magnitude* instead of the sign function as [21]

$$\tilde{S}_{(\mathbf{q}_m)}(i, j) \approx g(\mathbf{x}_j, \mathbf{q}_m) - g(\mathbf{x}_i, \mathbf{q}_m). \tag{7}$$

Then the new loss function becomes

$$L(g, \mathcal{X}, \mathcal{Y}) = -\sum_{m,i,j} [g(\mathbf{x}_j, \mathbf{q}_m) - g(\mathbf{x}_i, \mathbf{q}_m)] S_{(\mathbf{q}_m)}(i, j). \tag{8}$$

Intuitively, minimizing the above new loss function will not only maintain the triplet relationships, i.e., the ranking order between each pair of samples, but also impose a large margin constraint.

4. Ranking-Based Supervised Hashing

This section first introduces the form of hash function that is used in our approach. Then a critical step of converting binary hash codes to rank triplet is described. Finally the loss function using listwise supervision is defined, followed by an efficient solution using the augmented Lagrange multipliers (ALM) method. For simplicity, in this paper we use linear form in the proposed hash functions. However, our approach can be easily extended to learn nonlinear hash functions.

4.1. Linear Hash Functions

For a data point $\mathbf{x} \in \mathbb{R}^D$, a hash function $h(\cdot)$ is used to generate a binary code $h : \mathbb{R} \mapsto \{-1, 1\}$.[1] Linear hash functions, such as the well-known locality sensitive hashing [5], are computationally very efficient for large scale applications. Many recent learning based hashing techniques such as the semi-supervised hashing approach [21] applied this type of linear formulation. Assume the data samples are already zero-centered and the mean partition is used. Then we follow the general idea of linear functions to define our hash functions in the form as $h_k(\mathbf{x}) = \operatorname{sgn}\left(\mathbf{a}_\mathbf{k}^\top \mathbf{x}\right)$, where the coefficient $\mathbf{a}_k \in \mathbb{R}^D$ is a linear vector projecting the sample $\mathbf{x}$ to a one-dimensional space. Let $H = [h_1, \cdots, h_k, \cdots, h_K]$ be a sequence of hash functions. Then we can compute a K-bit binary code $H(\mathbf{x}) \in \mathbb{B}^\mathbf{K}$ for $\mathbf{x}$ as

$$H(\mathbf{x}) = [\mathbf{h_1}(\mathbf{x}), \cdots, \mathbf{h_k}(\mathbf{x}), \cdots, \mathbf{h_K}(\mathbf{x})]^\top = \operatorname{sgn}(\mathbf{A}^\top \mathbf{x})$$

where $\mathbf{A} = [\mathbf{a}_1, \cdots, \mathbf{a}_k, \cdots, \mathbf{a}_K] \in \mathbb{R}^{D \times K}$ is the coefficient matrix and $\mathbf{a}_k$ is the coefficient vector of the hash function $h_k(\cdot)$.

4.2. Deriving Rank Triplet from Hamming Embedding

Given a query $\mathbf{q}_m$ and the dataset $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$, the corresponding hash codes can be computed as $H(\mathbf{q}_m)$

[1] Here we generate hash bits as $\{-1, 1\}$, which are straightforward to convert to $\{0, 1\}$ valued hash codes.

and $H(\mathcal{X}) = [H(\mathbf{x}_1), \cdots, H(\mathbf{x}_N)]$. Then the Hamming distance can be used to rank the samples $\{\mathbf{x}_n\} \in \mathcal{X}$ associated the query sample $\mathbf{q}_m$, resulting in a ranking list $r_H\left(H(\mathbf{q}_m), H(\mathcal{X})\right)$. To represent such ranking list by the rank triplets, we first define the following similarity measure $g_H\left(H(\mathbf{q}_m), H(\mathbf{x}_i)\right)$ between hash codes using the normalized inner product, i.e., cosine similarity as

$$g_H\left(H(\mathbf{q}_m), H(\mathbf{x}_i)\right) = \frac{H(\mathbf{q}_m)^\top H(\mathbf{x}_i)}{\|H(\mathbf{q}_m)\| \cdot \|H(\mathbf{x}_i)\|}$$
$$= \frac{1}{K} H(\mathbf{q}_m)^\top H(\mathbf{x}_i) = \frac{1}{K} \sum_k h_k(\mathbf{q}_m) \cdot h_k(\mathbf{x}_i)$$

Following the computation from similarity measure to approximated ranking triplet in Eq.(7), we can map the Hamming embedding based similarity measure to

$$\tilde{S}_{(\mathbf{q}_m)}(i,j) \approx g_H\left(H(\mathbf{q}_m), H(\mathbf{x}_i)\right) - g_H\left(H(\mathbf{q}_m), H(\mathbf{x}_j)\right)$$
$$= \frac{1}{K} H(\mathbf{q}_m)^\top \left[H(\mathbf{x}_i) - H(\mathbf{x}_j)\right]. \quad (9)$$

It is easy to see that Hamming distance based ranking in the ascending order is equivalent to the ranking generated from the cosine similarity of hash codes in the descending order. Therefore, the corresponding ranking triplets $\tilde{S}_{(\mathbf{q}_m)}(i,j)$ are the same.

4.3. Final Objective and Optimization Method

Given the loss function definition in Eq.(8) and the derived ranking triplets in Eq.(9), now we neglect the constant $\frac{1}{K}$ and rewrite the loss function as below

$$L_H = -\sum_m \sum_{i,j} H(\mathbf{q}_m)^\top \left[H(\mathbf{x}_i) - H(\mathbf{x}_j)\right] S_{mij}$$

where the tensor element $S_{mij} = S_{(\mathbf{q}_m)}(i,j)$ is defined earlier. Note that the above loss function is still non-differentiable due to the embedded sign function. As suggested in [21], we drop off the sign function and use the signed magnitude in the loss function as

$$L_H = -\textstyle\sum_m \sum_{i,j} \mathbf{q}_m^\top \mathbf{A}\mathbf{A}^\top \left[\mathbf{x}_i - \mathbf{x}_j\right] S_{mij}$$
$$= -\textstyle\sum_m \mathbf{q}_m^\top \mathbf{A}\mathbf{A}^\top \mathbf{p}_m = -\operatorname{tr}(\mathbf{A}\mathbf{A}^\top \mathbf{B})$$

where $\mathbf{p}_m = \sum_{i,j} \left[\mathbf{x}_i - \mathbf{x}_j\right] S_{mij}$, $\mathbf{B} = \sum_m \mathbf{p}_m \mathbf{q}_m^\top$ and the symbol $\bullet$ represents Frobenius inner product. Note that $\mathbf{B}$ is computed using the training data $\{\mathbf{q}_m\}, \{\mathbf{x}_i\}$ and the ground truth triplet tensor $\mathbf{S}$. Although $\mathbf{B}$ is time consuming to calculate, it can be obtained off-line and remains as a constant during the optimization procedure. Maximizing the above objective function will provide an optimal solution of $\mathbf{A}$ which tends to preserve the given ranking orders.

Note that the above objective function has a similar expression as the objective for similarity function learning [1], which uses a non-smooth cost to include a margin-based ranking loss. Compared to Norouzi's objective of minimizing a hinge loss based cost [15], we formulate a smooth quadratic loss after a proper relaxation. Finally, we explicitly make the learned hash codes hold least redundant information by enforcing the orthogonality constraints. Following the work in [24, 21], we relax such hard constraints on hash codes and instead make the projection directions orthogonal, leading to the following:

$$\mathbf{A}^* = \arg\min_{\mathbf{A}} L_H = \arg\min_{\mathbf{A}} -\operatorname{tr}(\mathbf{A}\mathbf{A}^\top \mathbf{B}) \quad (10)$$
$$\text{s.t.} \quad \mathbf{A}^\top \mathbf{A} = \mathbf{I}.$$

A general method for solving the above constrained optimization problem is introduced as the augmented Lagrangian multiplier method (ALM). Such method has robust convergence performance and is well studied in the literature [2]. The augmented Lagrangian (AL) function is defined as

$$F(\mathbf{A}, \mathbf{\Lambda}) = -\operatorname{tr}(\mathbf{A}\mathbf{A}^\top \mathbf{B}) + \operatorname{tr}(\mathbf{\Lambda}(\mathbf{A}^\top \mathbf{A} - \mathbf{I}))$$
$$+ \frac{\rho}{2} \|\mathbf{A}^\top \mathbf{A} - \mathbf{I}\|_F^2, \quad (11)$$

where $\mathbf{\Lambda} \in \mathbb{R}^{K \times K}$ is the Lagrange multiplier matrix.

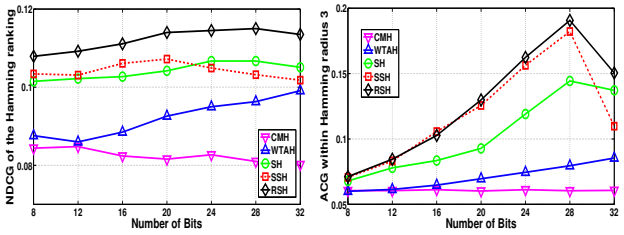
The ALM method is a primal-dual algorithm, as shown in the algorithm chart. Each iteration is composed of two steps. The first step updates the primal variable $\mathbf{A}$ for a fixed Lagrangian multiplier $\mathbf{\Lambda}_\tau$, by minimizing the AL function in Eq.(11). The second step updates the Lagrangian multiplier (the dual variable) along the gradient ascent direction $\nabla_{\mathbf{\Lambda}} F(\mathbf{A}, \mathbf{\Lambda})$. The primal step is solved using a simple gradient descent method, where the derivation of the gradient $\nabla_{\mathbf{A}} F(\mathbf{A}, \mathbf{\Lambda})$ is provided in the Appendix. The gradient descent method only requires first-order gradient information, which makes it very suitable for large scale problems. In comparison, Newton-type algorithms may have faster local convergence rate, however they also require much heavier computation in each iteration.

Algorithm 1 Using augmented Lagrange multipliers (ALM) method to minimize Eq.(11)

Input: constant matrix $\mathbf{B}$, initial $\mathbf{A}$, the coefficient $\rho > 0$, and the iteration count $\tau = 1$
while not converged **do**
  $\mathbf{A}_{\tau+1} = \arg\min F(\mathbf{A}, \mathbf{\Lambda}_\tau)$
  $\mathbf{\Lambda}_{\tau+1} = \mathbf{\Lambda}_\tau + \rho\left(\mathbf{A}_{\tau+1}^\top \mathbf{A}_{\tau+1} - \mathbf{I}\right)$
  Update the iteration count: $\tau = \tau + 1$
end while

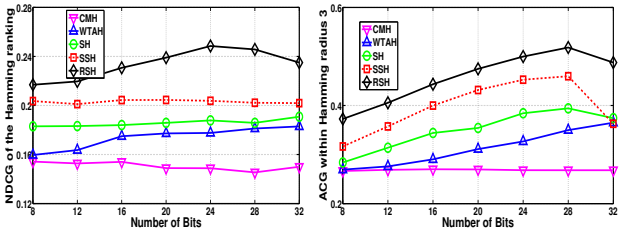
4.4. Complexity Analysis

The computational cost for learning the ranking-based supervised hash function consists of two parts, the offline



(a) (b)

Figure 2. Performance evaluation on **CIFAR** dataset using different number of hash bits. a) NDCG of the Hamming ranking; b) ACG within Hamming radius 3.



(a) (b)

Figure 3. Performance evaluation on **NUSWIDE** dataset using different number of hash bits. a) NDCG of the Hamming ranking; b) ACG within Hamming radius 3.

and online part. The offline calculation is mainly for deriving the constant matrix $\mathbf{B}$. Recall the definition of $\mathbf{p}_m$ and $\mathbf{B}$ described earlier, the complexity for computing the vector $\mathbf{p}_m$ is $O(DN^2)$ and for calculating $\mathbf{S}$ is $O(MD^2)$. Although it is time-consuming to obtain, the computation can be easily parallelized across different query samples. The online training for deriving the optimal hash functions is fairly fast. Notice that the AL function (11) is a nonconvex quadratic polynomial function in the entries of $\mathbf{A}$. Due to its structure, the gradient descent method converges to a local minimum starting from an initial point that is close enough to the optimal solution, which is often observed as a consistent and fast convergence behavior in practice. Each iteration of the gradient descent method involves matrix multiplications (see the Appendix). Their complexity can be counted: $\mathbf{BA}$ requires $O(KD^2)$ flops, $\mathbf{A\Lambda}$ requires $O(K^2D)$ flops, and constructing $\partial g(\mathbf{A})/\partial \mathbf{A}$ also requires $O(K^2D)$ flops (the main part of the computation is taken by $\mathbf{A}^\top\mathbf{A}$). Thus, the complexity of each iteration of the gradient descent method is $O(KD^2 + K^2D)$. Similarly, the dual update step has arithmetic complexity $O(K^2D)$. Hence, this algorithm is fairly scalable since its time complexity only relies on the feature dimensionality D and the number of bits K.

5. Experiments

We now apply the RSH technique to three image benchmark datasets i.e., **CIFAR**, **NUSWIDE**, and **One-Million** tiny images, which have been popularly adopted in the evaluation of hashing methods [6, 13, 21]. Extensive comparative studies with state-of-the-art hashing methods are also provided below.

5.1. Datasets

The **CIFAR** data contains a total of $60,000$ 32×32 color images with clean manual labels [8]. In particular, two levels of semantic class labels, i.e., super class and fine class, are assigned to the images. It has 20 superclass, each of which can be further split into 5 specific classes. It is intuitive to see that two images sharing a common fine class labels are more related than those sharing just a common superclass label, while the latter is apparently closer than those sharing no common class labels. Accordingly, we can derive ground-truth *weak* ranking lists with three relevance levels. The original intensity values are used as image presentations, resulting in a 3072-d feature vector.

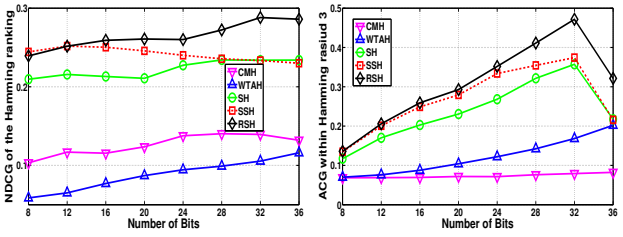
The **NUSWIDE** dataset is a set of Flickr images [3]. It has around 270K images manually annotated with 81 classes. Since each image in the **NUSWIDE** dataset is associated with multiple semantic classes, the semantic relevance can be easily derived based on the number of shared class labels. Compared to **CIFAR** dataset, **NUSWIDE** has much more relevance levels in the ranking lists. For feature representation, similar as [21], ℓ_2 normalized 1024-d sparse-coded bag-of-words descriptors are used.

The third dataset contains **One-Million** tiny images [20], each of which is represented by a 384-d GIST descriptor. This dataset has no semantic labels. Therefore we define the ground-truth relevance using the ℓ_2 distance of image features. For any query image, an image is recognized as strongly relevant if its distance to the query image is within the 2nd percentile of the whole set of distances. If the distance is beyond the 2nd percentile but within the 5th percentile, it is treated as weakly relevant. Otherwise, it is treated as an irrelevant image. Finally we can obtain ground-truth ranking lists with three relevance levels.

5.2. Experimental Setup

We compare the proposed *RSH* methods with four representative techniques, including spectral hashing (SH) [24], semi-supervised hashing (SSH) [21], and two order statistics based methods, i.e., concomitant min hashing (CMH) [4] and winner-takes-all hashing (WTAH) [27].

Since SH is a data-dependent method and does not use any label information, we use the standard settings in our experiments. For SSH, we randomly sample 1000 data points and use their ground-truth labels to generate pairwise similarity matrix as part of the training data. Similarly, for RSH, we randomly sample 100 query samples and 1000 data points to compute the ground-truth ranking lists, which finally give us a *triplet tensor* $\mathbf{S}$ with the size



(a) (b)

Figure 4. Performance evaluation on **One-Million** tiny image dataset using different number of hash bits. a) NDCG of the Hamming ranking; b) ACG within Hamming radius 3.

$100 \times 1000 \times 1000$. Although the supervision information used in RSH is much richer than SSH, the required labeled training samples for RSH (100 training queries plus 1000 database points) are only slightly more than that used in SSH. For both CMH and WTAH, we use the best settings reported in the literatures [4][27]. Finally, in the test stage, we use 1000 random samples as queries and evaluate the quality of the returned samples.

5.3. Evaluation Metrics

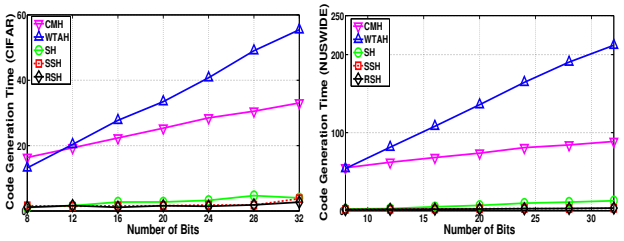
Following the evaluation method suggested in [23], we measure the search quality of a *single* hash table using both *hamming ranking* and *hash lookup*. For *hamming ranking*, the database points are ranked based on the Hamming distance to the query point. We use normalized discounted cumulative gain (NDCG) to evaluate the ranking quality in Hamming space for each individual query [7], which is a very popular measure in the IR community. Discounted cumulative gain (DCG) uses a graded relevance scale to measure the effectiveness of a search algorithm:

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}. \quad (12)$$

Here p indicates the truncated position in a ranking list and the value of rel_i indicates the relevance level for the returned ith sample. It is easy to see that the graded relevance value of each returned sample is reduced logarithmically proportional to its position. Accordingly, if the ideal ranking gives the DCG value as $IDCG$, NDCG is calculated as $NDCG_p = \frac{DCG_p}{IDCG_p}$. For those samples falling in the same Hamming distance to query points, the expectation of NDCG is computed.

On the other hand, *hash lookup* returns the samples within a certain Hamming radius r (set as $r \leq 3$ in the experiments). Since hash lookup does not provide ranking for returned points with equal Hamming distance to the queries, we use average cumulative gain (ACG) to measure the quality of these returned samples [7], which is calculated as

$$ACG_r = \frac{1}{|\mathcal{N}_r|} \sum_{\mathbf{x} \in \mathcal{N}_r} rel_{\mathbf{x}}. \quad (13)$$



(a) (b)

Figure 5. Code generation time of different methods on the **CIFAR** and **NUSWIDE** datasets.

Here $\mathcal{N}_r$ denotes the returned data points within a Hamming radius r and $rel_{\mathbf{x}}$ is the relevance level of a returned data point $\mathbf{x}$. Here the scale value of $|\mathcal{N}_r|$ is the total number of returned data points. Hence, ACG essentially measures the average precision weighted by the relevance level of each returned sample. In summary, both metrics emphasize the quality of ranking, which is practically important.

5.4. Results

The performances using various numbers of hash bits are presented in Figures 2, 3, 4, for the **CIFAR**, **NUSWIDE** and **One-Million** datasets, respectively. Clearly, RSH outperforms the compared methods in most of the tested cases. Particularly, for the evaluations by NDCG over the the Hamming ranking in Figures 2(a), 3(a), 4(a), RSH achieves significant performance gains over all the other methods. This clearly demonstrates that RSH tends to preserve the ranking order in the learned Hamming space. For performance comparison of hash lookup performance, SSH achieves a similar high performance, and even outperforms RSH in a few tested cases on the **CIFAR** dataset and the **One-Million** dataset. Note that **CIFAR** dataset and the **One-Million** dataset only have three relevance levels which encode very weak ranking orders. When using longer bits, the Hamming embedding becomes increasingly sparse and many queries have empty returns (treated as zero cumulative gain), which prevents a single hash table to achieve higher performance. The partial order statistics based methods, i.e., WTAH and CMH, do not perform well over these challenging high-dimensional datasets.

The time cost for training the hash functions can be ranked as : $RSH > SSH > SH > WTAH \simeq CMH$. It requires around several hundreds to a thousand seconds to train RSH function with 8-bit to 32-bit, about 2-5 times of that of SSH, which is actually not slow considering the complexity of training. In contrast to the offline training, the online code generation time is more critical for real-world search applications. Figure 5 shows the time cost for generating hash codes using different approaches on **CIFAR** and **NUSWIDE**. RSH and SSH are the most efficient in terms of code generation since they only need linear projection and binarization. SH requires a little more time due to the

sinusoidal binarization process. WTAH and CMH take the longest time to compute the hash codes since both of them need a sorting process over the feature space. Especially, WTAH is fairly slow in generating long hash codes.

6. Conclusions

In this paper, we have introduced a learning to hash framework through leveraging *listwise supervision* to train efficient hash functions. Our approach holds the following novel aspects. First, realizing the difficulty of directly optimizing over discrete ranking orders, we introduced a triplet representation for listwise supervision and proved that this representation is equivalent to rank orders. Then, we proposed to match Hamming ranking to the given ranking in the semantic space through minimizing the inconsistency between two triplet representations. Finally, to solve the above objective, we proposed an efficient solution by using the augmented Lagrangian multiplier method.

We performed extensive experiments on three large image datasets and compared with the state-of-the-art hashing techniques. Experimental results demonstrated that the proposed ranking-based supervised hashing method yields superior performance. In addition, since the proposed method uses linear functions, its online code generation time is extremely fast, especially when compared with those order statistics based methods. Important future works include the extension to nonlinear cases by applying kernels and the design of multiple ranking-based supervised hash tables to further boost image search quality.

References

[1] B. Bai, J. Weston, D. Grangier, R. Collobert, K. Sadamasa, Y. Qi, O. Chapelle, and K. Weinberger. Supervised semantic indexing. In *Proc. of the 18th ACM CIKM*, pages 187–196, 2009.

[2] D. Bertsekas. Constrained optimization and lagrange multiplier methods. *Computer Science and Applied Mathematics, Boston: Academic Press*, 1, 1982.

[3] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng. Nus-wide: A real-world web image database from national university of singapore. In *Proc. of ACM CIVR*, Santorini, Greece, July 2009.

[4] K. Eshghi and S. Rajaram. Locality sensitive hash functions based on concomitant rank order statistics. In *Proc. of ACM SIGKDD*, pages 221–229, 2008.

[5] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. of VLDB*, pages 518–529, 1999.

[6] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proc. of IEEE CVPR*, pages 817–824, 2011.

[7] K. Järvelin and J. Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *Proc. of ACM SIGIR*, pages 41–48, 2000.

[8] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

[9] B. Kulis and T. Darrell. Learning to Hash with Binary Reconstructive Embeddings. In *Proc. of NIPS*, volume 22, pages 1042–1050, 2009.

[10] B. Kulis, P. Jain, and K. Grauman. Fast similarity search for learned metrics. *IEEE TPAMI*, 31(12):2143–2157, 2009.

[11] H. Li. A short introduction to learning to rank. *IEICE Trans. on Information and Systems*, 94(10):1854–1862, 2011.

[12] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *Proc. of IEEE CVPR*, Providence, USA, 2012.

[13] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *Proc. of ICML*, Bellevue, USA, 2011.

[14] M. Norouzi and D. Fleet. Minimal loss hashing for compact binary codes. In *Proc. of ICML*, Bellevue, USA, 2011.

[15] M. Norouzi, D. Fleet, and R. Salakhutdinov. Hamming distance metric learning. In *Proc. of NIPS*, volume 25, pages 1070–1078, 2012.

[16] P. Ram, D. Lee, H. Ouyang, and A. Gray. Rank-approximate nearest neighbor search: Retaining meaning and speed in high dimensions. In *Proc. of NIPS*, volume 22, pages 1536–1544, 2009.

[17] R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[18] G. Shakhnarovich. *Learning task-specific similarity*. PhD thesis, Massachusetts Institute of Technology, 2005.

[19] A. Shashua and A. Levin. Ranking with large margin principle: Two approaches. In *Proc. of NIPS*, volume 16, pages 937–944, 2003.

[20] A. Torralba, R. Fergus, and W. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE TPAMI*, 30(11):1958–1970, 2008.

[21] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *Proc. of IEEE CVPR*, pages 3424–3431, San Francisco, USA, June 2010.

[22] J. Wang, S. Kumar, and S.-F. Chang. Sequential projection learning for hashing with compact codes. In *Proc. of ICML*, pages 1127–1134, Haifa, Israel, 2010.

[23] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large-scale search. *IEEE Trans. on PAMI*, 34(12):2393–2406, 2012.

[24] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proc. of NIPS*, volume 21, pages 1753–1760, 2008.

[25] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. In *Proc. of ICML*, pages 1192–1199, 2008.

[26] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu. Complementary hashing for approximate nearest neighbor search. In *Proc. of IEEE ICCV*, pages 1631–1638, 2011.

[27] J. Yagnik, D. Strelow, D. Ross, and R.-S. Lin. The power of comparative reasoning. In *Proc. of IEEE ICCV*, pages 2431–2438, 2011.