Natural Supervised Hashing

Qi Liu, Hongtao Lu*
Key Laboratory of Shanghai Education Commission
for Intelligent Interaction and Cognitive Engineering
Department of Computer Science and Engineering,
Shanghai Jiao Tong University, P.R. China
luoguliu@gmail.com, lu-ht@cs.sjtu.edu.cn

Abstract

Among learning-based hashing methods, supervised hashing tries to find hash codes which preserve semantic similarities of original data. Recent years have witnessed much efforts devoted to design objective functions and optimization methods for supervised hashing learning, in order to improve search accuracy and reduce training cost. In this paper, we propose a very straightforward supervised hashing algorithm and demonstrate its superiority over several state-of-the-art methods. The key idea of our approach is to treat label vectors as binary codes and to learn target codes which have similar structure to label vectors. To circumvent direct optimization on large $n \times n$ Gram matrices, we identify an inner-product-preserving transformation and use it to bring close label vectors and hash codes without changing the structure. The optimization process is very efficient and scales well. In our experiment, training 16-bit and 96-bit code on NUS-WIDE cost respectively only 3 and 6 minutes.

1 Introduction

Hashing has recently attracted considerable research efforts from machine learning, computer vision, information retrieval and related areas. Given a dataset, hashing methods generate binary codes that intend to preserve metric distances or semantic similarities of the original data. Due to the efficient computation of Hamming distance between binary codes, hashing is very suitable for large-scale ANN (approximate nearest neighbor) search. Compact hash codes can produce significant gains in both memory usage and search efficiency, maintaining reasonable accuracies at the same time.

Locality-Sensitive Hashing (LSH) [Gionis *et al.*, 1999] is a well-known *data-independent* hashing method, which uses random projections to construct hash functions. It has continued to be a research interest in the past decade and efforts have been devoted to expand LSH to more distance metrics including l_p distance [Datar *et al.*, 2004] and kernel distance [Kulis and Grauman, 2009]. Due to theoretical guarantees that the original similarity is asymptotically preserved, LSH is a rather popular method in ANN search and usually long codes are used to obtain a good search precision. However, long codes would result in low recall. So in practical settings, LSH also requires multiple hash tables to achieve reasonable recall rate. Both long codes and multiple tables will increase memory usage and reduce search efficiency.

Unlike *data-independent* hashing which uses random prjections, *data-dependent* methods try to learn hash functions from training data. *Data-dependent* hashing can produce more compact binary codes and can usually achieve better accuracy even with much shorter codes. Due to this property, researchers have been putting a lot of efforts on designing good hash codes. *Data-dependent* hashing can be roughly divided into three categories: *unsupervised*, *semi-supervised* and *supervised hashing*.

Unsupervised hashing uses unlabeled data to train a set of hash functions, aiming at preserving the metric neighborhood (usually, l_2 metric) of the original data. The representative of unsupervised hashing algorithms includes Spectral Hashing (SH) [Weiss *et al.*, 2009], Iterative Quantization (ITQ) [Gong and Lazebnik, 2011], Isotropic Hashing (IsoH) [Kong and Li, 2012], Harmonious Hashing (HamH) [Xu *et al.*, 2013] e.t.c. Among these methods, ITQ uses PCA projections as initialization, and then tries to find an orthogonal matrix which minimizes the quantization loss between binary codes and rotated points. Due to the structure-preserving property of rotation and minimization of quantization loss, ITQ has achieved state-of-the-art performance in unsupervised hashing methods. Recently, the idea of ITQ is also used in producing very long codes for high-dimensional features [Gong *et al.*, 2013; Xia *et al.*, 2015] and has shown promising results.

However, metric neighbors are not necessarily semantic neighbors. While unsupervised hashing achieves promising performance in preserving metric distances, in some applications we want ANN search to return semantic neighbors for a given query. This leads to (semi-)supervised hashing methods. In this paper, we focus on formulation and optimization of supervised hashing.

In contrast to unsupervised hashing whose training data is unlabeled, (semi-)supervised methods use labeled training data to learn semantic-similarity-preserving codes. The difference between semi-supervised and supervised hashing is that supervised methods use pure labeled data, while semi-

*corresponding author

supervised hashing deals with the situation where only part of training data is labeled. The representatives of (semi-)supervised hashing include Semi-Supervised Hashing (SSH) [Wang *et al.*, 2012], Sequential Projection Learning Hashing (SPLH) [Wang *et al.*, 2010], Binary Reconstructive Embedding (BRE) [Kulis and Darrell, 2009], Minimal Loss Hashing (MLH) [Norouzi and Blei, 2011], CCA-ITQ [Gong and Lazebnik, 2011], Kernel-based Supervised Hashing (KSH) [Liu *et al.*, 2012], Ranking-based Supervised Hashing (RSH) [Wang *et al.*, 2013], Graph Cut Coding (GCC) [Ge *et al.*, 2014], Fast Hashing (FastH) [Lin *et al.*, 2014], Supervised Discrete Hashing (SDH) [Shen *et al.*, 2015] e.t.c. Research efforts have been devoted to several aspects in the literature, which we briefly discuss in the following text.

The first thing to consider is to define some criteria which express desired properties of target codes and guide the design of objective functions. Different algorithms are essentially different from each other on the criterion they are built upon. For example, BRE [Kulis and Darrell, 2009] utilizes pairwise similarity and intends to find binary codes which generate small Hamming distance for similar pairs and large distance for dissimilar ones. The idea of pairwise similarity is subsequently used by other methods (SSH, SPLH, KSH, GCC, FastH) and is now the most popular supervision form in the literature. Different from these approaches, CCA-ITQ and SDH utilize semantic labels of individual samples instead of pairwise similarity. To emphasize the difference, we call them *pointwise* in this paper. CCA-ITQ consists of two steps. It first uses CCA to find a common space for original features and labels such that in this space embeddings of features and labels have maximum correlation. Then a ITQ process is applied on the embedding of original feature and a rotation matrix is learned to minimize quantization loss just like the unsupervised case. CCA-ITQ relies on the CCA step to exploit supervised information and the correlation maximization is the key criterion. In contrast, the recently proposed SDH is more direct, which requires learnt binary codes to be good features for linear classification with given labels. And due to direct focus on binary codes and carefully designed optimization process, SDH has been reported to achieve state-of-the-art results [Shen *et al.*, 2015]. In this paper, we propose a new pointwise method which attempts to generate binary codes with close structure to label vectors. We show that our approach is even more direct than SDH and demonstrate its superiority on several datasets.

More complex hash functions are also an interest in the field. The most common hash function is linear projection followed by thresholding, which has been used by many methods. Afterwards, to deal with the situation where samples are not linearly separable, hash functions with kernels are proposed [Kulis and Darrell, 2009; Liu *et al.*, 2012]. Kernelization has been proved able to boost the performance of supervised hashing methods, especially when we are using weak features. Recently, hashing methods based on deep neural networks are quite popular [Xia *et al.*, 2014; Erin Liong *et al.*, 2015]. These approaches have shown great improvement in performance since deep networks can learn much more complex nonlinear functions than a single kernel. In this paper, we choose linear projection and the kernelized version as hash functions. We do not use deep networks because we want to focus on the criterion of hash codes, i.e. design of loss functions. We think that our method can easily accommodate deep networks by changing hash functions to neural networks and modifying the update of projection matrix to update of network weights with back propagation.

In this paper, we investigate label vectors as binary codes for semantic similarity search. Inspired by its good performance, we try to learn hash codes with similar structure to label vectors. Because it is *natural* to use label vectors to encode semantic information, we dub the proposed approach *Natural Supervised Hashing*. Our contributions are as follows:

1. We formulate a novel objective function to learn hash codes which produce close inner products to label vectors. And to avoid direct optimization on big $n \times n$ Gram matrices, we identify an structure-preserving transformation to fill the dimensionality gap between target codes and label vectors. With this transformation, we could implicitly bring close inner products of hash codes and that of label vectors without explicit computation of Gram matrices.
2. We propose an alternating optimization method for our final objective, with each step having an analytical solution. The training process is very fast and scales well. On NUS-WIDE, training 16-bit and 96-bit code cost respectively only 3 and 6 minutes.

The remainder of the paper is organized as follows: section 2 presents the proposed approach and section 3 demonstrates the performance of our method compared with several state-of-the-art algorithms; we conclude the paper in section 4.

2 Natural Supervised Hashing

In this section, we present the objective function and learning algorithm.

2.1 Label vectors as binary codes

For dataset with l labels, the *ideal* way to encode x_i for semantic search is using its label vector $y_i \in \{0, 1\}^l$ as binary code. The j-th element in y_i equals to 1 if sample i has label j, and is 0 otherwise. For single-label datasets, only one element in y_i is 1 while there can be multiple 1s in multi-label cases. This simple code makes sense because the Hamming distance between each pair is just the number of different labels, which we call *semantic distance*.

However, we find that although labels are ideal to measure difference of semantic information between samples, it is *not* necessarily optimal in NN search[1]. This problem stems from inconsistencies between semantic distance and evaluation criteria. In evaluation we decide whether a pair is similar by counting the number of labels they share, i.e. inner product between labels. Due to this criterion, two samples with large semantic distance may be considered similar in experiments. For example, if we have $y_i = [1, 1, 1, 0]^T$ and $y_j = [0, 1, 1, 1]^T$, we tend to determine sample i and sample j as similar because they share 2 labels when there are

[1]It is indeed optimal for certain cases, like single-label datasets.

only 4. However the Hamming distance between i and j is $D_h(y_i, y_j) = 2$, which is not small in 4-bit code.

We believe the suboptimality is a problem with the evaluation criteria commonly adopted, not the label vectors, considering they are ground-truth semantic annotations. In fact, despite this problem, labels still give very good retrieval accuracy. On NUS-WIDE, label vectors achieve a 0.772 mAP within top-5K retrieved samples, much higher than results obtained by state-of-the-art supervised hashing methods.

Inspired by its good performance, we intend to formulate an objective function and learn hash code from binary labels. Note that while label vector is of fixed length, our desired hash code may have varying number of bits.

2.2 Preserving inner product

We want our code to generate close Hamming distance to labels. Inspired by KSH [Liu *et al.*, 2012], we do not directly work on Hamming distance and try to manipulate code's inner product instead. Unlike KSH which uses $-1/1$ bit value during training, we use $0/1$ bit for simplicity.

Intuitively, to make inner product of labels and of target code as close as possible, we can write the objective as follows:

$$L(P) = ||YY^T - \phi(XP)\phi(XP)^T||_F^2 \tag{1}$$

where $X \in \mathbb{R}^{n\times d}$ is training data, $Y \in \{0,1\}^{n\times l}$ is label matrix, and $P \in \mathbb{R}^{d\times b}$ contains project directions to learn. $\phi(\cdot)$ is an elementwise function with $\phi(x) = 1$ for $x > 0$ and $\phi(x) = 0$ otherwise. This objective seeks directions that bring close two codes' inner product. However, we find it very hard to optimize due to the presence of $\phi(XP)\phi(XP)^T$. Besides, the two large $n\times n$ matrices would take a lot of memory (for $n = 50000$, YY^T needs about 9.3 GB memory with *single* type). So for some large datasets, e.g. NUS-WIDE, which contains $269,648$ samples in total, we cannot utilize full training data in learning process.

To resolve these difficulties, we try to find a inner-product-preserving transformation, with which we can achieve the goal without explicit computation of Gram matrices. We observe that if a matrix $R \in \mathbb{R}^{l\times m}$ has orthonormal rows, i.e. $RR^T = I_l$, then we can get

$$YR(YR)^T = YRR^TY^T = YY^T \in \mathbb{R}^{n\times n} \tag{2}$$

i.e. YR generates the same Gram matrix as Y. The transformation R preserves Y's structure. So besides minimizing the distance between $\phi(XP)\phi(XP)^T$ and YY^T explicitly, we could minimize the distance between $\phi(XP)$ and YR instead, which brings close two Gram matrices implicitly.

The condition $RR^T = I_l$ forces a dimensionality constraint: there should be less rows than columns in R. Due to this constraint, our final optimization problem has two cases:

- For $b \leqslant l$,

$$\begin{aligned} \min_{P,R} \quad & ||Y - \phi(XP)R||_F^2 \\ \text{s.t.} \quad & R \in \mathbb{R}^{b\times l}, \quad RR^T = I_b \end{aligned} \tag{3}$$

- For $l < b$,

$$\begin{aligned} \min_{P,R} \quad & ||YR - \phi(XP)||_F^2 \\ \text{s.t.} \quad & R \in \mathbb{R}^{l\times b}, \quad RR^T = I_l \end{aligned} \tag{4}$$

We use R to fill the dimensionality gap between label vectors and target codes, preserving inner product at the same time.

2.3 Optimization

Our goal is to learn project directions P. We optimize the objective function in an alternating way, i.e. fixing P, optimize R and fixing R, update P. For different cases, we have different iteration steps.

For $b \leqslant l$

The objective function can be expanded as follows:

$$\begin{aligned} & ||Y - \phi(XP)R||_F^2 \\ &= ||Y||_F^2 - 2tr(Y^T\phi(XP)R) + tr(R^T\phi(XP)^T\phi(XP)R) \\ &= -2tr(Y^T\phi(XP)R) + tr(\phi(XP)^T\phi(XP)) \\ &\quad + const \end{aligned} \tag{5}$$

where $const$ indicates a constant term that does not depend on either P or R.

- R step: Fixing P, the optimization problem of R is

$$\begin{aligned} \max_{R} \quad & tr(Y^T\phi(XP)R) \\ \text{s.t.} \quad & R \in \mathbb{R}^{b\times l}, \quad RR^T = I_b \end{aligned} \tag{6}$$

  This problem turns out to be a Procrustes problem with analytic solutions [Xia *et al.*, 2015]. First we perform SVD $Y^T\phi(XP) = U\Sigma V^T$, where U is an $l \times l$ orthogonal matrix, Σ is an $l \times b$ matrix and V is an $b \times b$ orthogonal matrix. Then the solution is $R = V\hat{U}^T$, where $\hat{U}$ contains first b columns of U.

- P step: The problem of P is hard to optimize since the ϕ function is nonsmooth. Following the idea of SSH [Wang *et al.*, 2012], we relax ϕ for easy optimization. The problem is now

$$\min_{P} \quad -2tr(RY^TXP) + tr(P^TX^TXP) \tag{7}$$

  After setting the gradient to zero, we get a closed-form solution

$$P = (X^TX)^{-1}X^TYR^T \tag{8}$$

  To prevent singularity, we can add a regularization term to the solution and obtain:

$$P = (X^TX + \lambda I_d)^{-1}X^TYR^T \tag{9}$$

For $l < b$

The second case is quite similar. The objective function can be written as

$$\begin{aligned} & ||YR - \phi(XP)||_F^2 \\ = \quad & -2tr(\phi(XP)^TYR) + tr(\phi(XP)^T\phi(XP)) \\ & + const \end{aligned} \tag{10}$$

The difference to the $b \leqslant l$ case is minor.

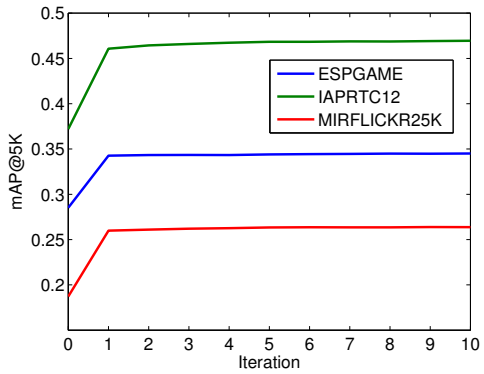


Figure 1: mAP@5K of NSH with 64 bits on three datasets. Our result improves along training and converges quickly. In fact, performance changes little after the first iteration.

- R step: For fixed P , the problem with respect to R reduces to

$$\begin{aligned} \max_{R} \quad & tr(\phi(XP)^T YR) \\ \text{s.t.} \quad & R \in \mathbb{R}^{l\times b}, \quad RR^T = I_l \end{aligned} \tag{11}$$

  which is also a Procrustes problem. We do SVD $\phi(XP)^T Y = U\Sigma V^T$. U is an $b \times b$ orthogonal matrix, Σ is an $b \times l$ matrix and V is an $l \times l$ orthogonal matrix. The solution is $R = V\hat{U}^T$, where $\hat{U}$ contains first l columns of U.

- P step: Again we relax ϕ and the optimization problem is:

$$\min_{P} -2tr(P^T X^T YR) + tr(P^T X^T XP) \tag{12}$$

  After adding a regularizer, we get the solution

$$P = (X^T X + \lambda I_d)^{-1} X^T YR \tag{13}$$

For both cases, alternating steps have analytic solutions, leading to fast optimization during training. Note that the term $A = (X^T X + \lambda I_d)^{-1} X^T Y$ can be computed only once before iterating, and then P is updated by $P = AR^T$ or $P = AR$.

Due to relaxation in P step, objective value may not decline after one iteration. Nevertheless, we find that search qualities do improve along iterations. Figure 1 shows mAP of 5K top-ranked samples on three datasets when we are training 64-bit code. Our result improves along iterations and converges quickly, showing the efficacy of the optimization process. The quick convergence is a result of strict constraints on R. In our experiments, we find that the two variables change little after first several iterations.

2.4 Kernelization

Kernlization is popular in supervised hashing literature. Given a kernel $\kappa(\cdot,\cdot)$, we randomly sample m ($m < n$) anchor points $\{x_1, x_2, ..., x_m\}$ from training data. Then original data $x \in \mathbb{R}^d$ can be transformed to $k(x) = [\kappa(x_1, x), \kappa(x_2, x), ..., \kappa(x_m, x)]^T \in \mathbb{R}^m$ and the kernelized data matrix is $K \in \mathbb{R}^{n\times m}$. Then we can learn hash functions in the new space $P \in \mathbb{R}^{m\times b}$. The optimization is exactly the same as above, only with $\phi(XP)$ replaced by $\phi(KP)$. In the experiments, we use RBF kernel because datasets are represented with weak features.

Dataset	database	query	label	anchor
NUS-WIDE	268648	1000	81	1000
IAPRTC12	17665	1962	291	1000
ESPGAME	18689	2081	268	1000
MIRFLICKR25K	24000	1000	38	500

Table 1: Statistics of the four datasets used in evaluation. We use less anchors for MIRFLICKR25K because it is not lage-scale and contains less labels than others.

Method	#training	16	32	64	96
LSH	268648	0.95	1.77	3.59	4.92
PCA-ITQ	268648	5.84	13.2	31.6	51.2
BREs	5000	2527	4610	10883	-
KSH	5000	1009	2057	4121	6334
CCA-ITQ	268648	26.1	33.8	50.7	129
SDH	268648	321	523	1234	2306
NSH	268648	182	218	284	338

Table 2: Training time on NUS-WIDE for learning different number of bits. All numbers are given in seconds. Our NSH scales well when bits increase.

3 Experiments

We evaluate our method on 4 datasets: NUS-WIDE IAPRTC12[2], ESPGAME [3], MIRFLICKR25K [4]. For NUS-WIDE [Chua *et al.*, 2009], we use the 500 dimensional bag-of-words vectors provided by the authors. For the other three, we use the 512 dimensional GIST features provided by [Guillaumin *et al.*, 2010]. Each dataset is split into a database and a query set. Statistics of datasets are given in table 1.

We compare the proposed NSH against six state-of-the-art hashing algorithms, including unsupervised LSH [Datar *et al.*, 2004], PCA-ITQ [Gong and Lazebnik, 2011], and supervised BREs [Kulis and Darrell, 2009], KSH [Liu *et al.*, 2012], CCA-ITQ [Gong and Lazebnik, 2011] and SDH [Shen *et al.*, 2015]. BREs indicates the supervised version of BRE. We kernelize the data for all *supervised* methods, including CCA-ITQ for fair comparion. The kernel we choose is RBF $\kappa(x, y) = exp(||x - y||/\sigma^2)$, where σ is set to an appropriate value. 1000 anchors are sampled from databases for all datasets except MIRFLICKR25K, for which we use only 500 anchors considering it is not large-scale and contains less labels.

In supervised methods, BREs and KSH construct a $n \times n$ pairwise similarity matrix to learn hash codes. This matrix takes much memory (9.3 GB for $n = 50000$) thus we cannot use all available labeled data for training. Besides, when n is

[2]http://www.imageclef.org/photodata
[3]http://www.hunch.net/~jl/
[4]http://press.liacs.nl/mirflickr/

Method	mAP@5K				precision@radius 2			recall@radius 2		
	16	32	64	96	16	32	64	16	32	64
LSH	0.1961	0.2038	0.2053	0.2123	0.1957	0.0621	0.0095	0.0052	0.0001	0
PCA-ITQ	0.2151	0.2298	0.2342	0.2329	0.2118	0.2148	0.0410	0.0275	0.0031	0.0013
l_2	0.2181				-			-		
BREs	0.2138	0.2279	0.2349	-	0.2135	0.2006	0.0301	0.0127	0.0006	0
KSH	0.2631	0.2778	0.2809	0.2809	0.2597	0.2644	0.1131	0.0362	0.0072	0.0004
CCA-ITQ	0.2787	0.2882	0.2952	0.2966	0.2740	0.2677	0.1117	0.0377	0.0042	0.0004
SDH	0.2820	0.2883	0.2924	0.2989	0.2721	0.2826	0.2004	0.0698	0.0177	0.0043
NSH	**0.2979**	**0.3109**	**0.3169**	**0.3216**	**0.2850**	**0.3014**	**0.2532**	0.1497	0.0597	0.0180
svm	0.1080				0.1274			**0.4023**		

Table 3: Experimental results on NUS-WIDE with varying number of bits. We perform both hamming ranking and hash lookup evaluation. Best results are in bold.

Method	mAP@5K on IAPRTC12					mAP@5K on ESPGAME				
	8	16	32	64	128	8	16	32	64	128
LSH	0.3253	0.3421	0.3585	0.3656	0.3815	0.2812	0.2775	0.2793	0.2897	0.2936
PCA-ITQ	0.3492	0.3652	0.3745	0.3811	0.3860	0.3017	0.2981	0.2983	0.2991	0.3000
l_2	0.3941					0.2964				
BREs	0.3475	0.3690	0.3903	0.4032	-	0.2954	0.3035	0.3082	0.3130	-
KSH	0.3819	0.4102	0.4280	0.4363	0.4445	0.3025	0.3218	0.3289	0.3335	0.3379
CCA-ITQ	0.3952	0.4106	0.4221	0.4280	0.4340	0.3297	0.3313	0.3335	0.3339	0.3359
SDH	0.3784	0.4038	0.4191	0.4296	0.4352	0.3171	0.3244	0.3276	0.3320	0.3350
NSH	**0.4028**	**0.4327**	**0.4490**	**0.4562**	**0.4668**	**0.3379**	**0.3400**	**0.3426**	**0.3462**	**0.3500**
svm	0.2135					0.1563				

Table 4: Experimental results on IAPRTC12 and ESPGAME with varying number of bits. Best results are in bold.

large, training time needed for BREs and KSH is prohibitive. Therefore, we randomly sample 5000 points from databases as training sets for BREs and KSH. In contrast, CCA-ITQ, SDH and our NSH have better scalability and are trained with the whole database. In experiments, we give NSH 50 training iterations and set $\lambda = 10^{-4}$ for all cases.

We also evaluate two baselines. One is direct linear search with original features, denoted as l_2. The other is training a linear SVM on kernelized data for each label. And when received a query, a binary code is first generated with trained SVMs, and then it is used to search through databases of labels. We choose Pegasos [Shalev-Shwartz *et al.*, 2011] as our SVM for its fast training speed and small memory consumption.

We follow two search procedures: *hamming ranking* and *hash lookup*. Hamming ranking measures search quality through ranking database points according to their Hamming distance to queries and calculating precision, recall and mean Average Precision (mAP) in top-ranked samples. Hash lookup finds database points within a given Hamming radius of queries and calculates precision and recall in retrieved samples. Note that we treat failing to find any samples for a query as zero precision. All our results are obtained on a laptop with Intel Core i5-3210M 2.50 GHz and 12 GB RAM.

3.1 Results on NUS-WIDE

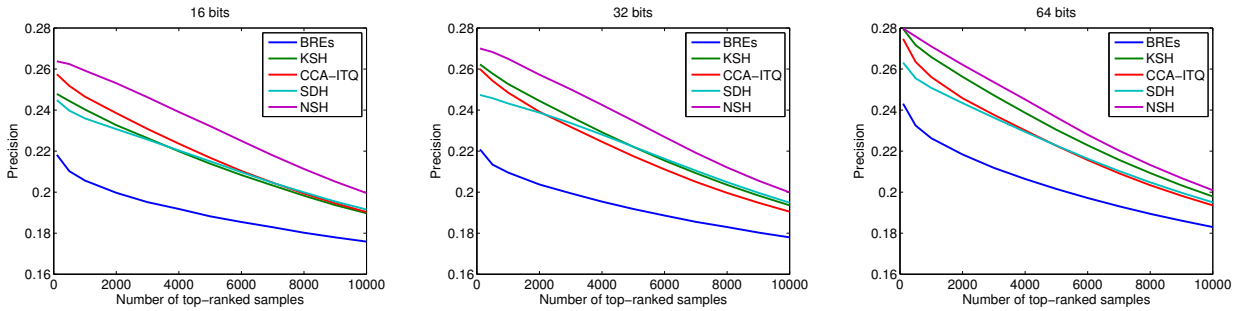
NUS-WIDE is sparsely labeled, with each sample associated with 1.87 labels on average. We choose NUS-WIDE for its large scale and determine a pair as ground-truth similar if they share at least one label. Training time of evaluated methods is given in table 2, from which we see that NSH costs much less time than BREs, KSH and it scales well when bits increase, while the training cost of SDH seems to be linear with bits. Evaluation results are presented in table 3. We see that NSH consistently attains best mAP and precision within radius 2 under different number of bits, and the margin between NSH and the second best SDH is not small. Considering recall within radius 2, NSH again outperforms other hashing methods. The baseline svm attains high $recall@2$ because SVMs are trained with much less positive samples than negative samples and tend to predict all zero binary codes, which has a small Hamming distance from sparse labels. This can be inferred from the low $precision@2$.

3.2 Results on IAPRTC12 and ESPGAME

IAPRTC12 and ESPGAME are also sparsely labeled, with each sample associated with respective 5.72 and 4.69 labels on average. We choose the two datasets because they contain much more labels than NUS-WIDE, 291 for IAPRTC12 and 268 for ESPGAME, leading to a more difficult problem. In evaluation, a pair is determined as similar if they share at least one label. Experimental results are given in table 4. We see that NSH consistently achieves best performance from 8 bits to 128 bits on both datasets, with improvements over other methods ranging from 2% to 5%. The baseline svm considers each label separately and does not take the whole structure

Method	mAP@5K				precision@radius 2			recall@radius 2		
	8	16	32	64	8	16	32	8	16	32
LSH	0.1786	0.1794	0.1895	0.1898	0.1723	0.1867	0.0926	0.1387	0.0107	0.0003
PCA-ITQ	0.1912	0.1925	0.1962	0.1987	0.1826	0.1910	0.0167	0.1638	0.0026	0.0001
l_2	0.1935				-			-		
BREs	0.1940	0.2012	0.2101	0.2199	0.1856	0.2094	0.0132	0.1478	0.0011	0
KSH	0.2250	0.2332	0.2494	0.2567	0.2110	0.2299	0.0506	0.1861	0.0045	0.0001
CCA-ITQ	0.2337	0.2391	0.2448	0.2485	0.2157	0.2543	0.1245	0.2060	0.0136	0.0019
SDH	0.2227	0.2385	0.2375	0.2443	0.2042	0.2344	0.0954	0.2449	0.0319	0.0003
NSH	**0.2447**	**0.2530**	**0.2599**	**0.2634**	**0.2161**	**0.2555**	**0.2140**	**0.2720**	**0.0510**	0.0160
svm	0.0749				0.0769			0.0245		

Table 5: Experimental results on MIRFLICKR25K with varying number of bits. We perform both hamming ranking and hash lookup evaluation. Best results are in bold.



(a) Precision with 16 bits (b) Precision with 32 bits (c) Precision with 64 bits

Figure 2: Hamming ranking precision of top-ranked samples on MIRFLICKR25K with varying number of bits.

into account, resulting in even worse performance than unsupervised hashing methods.

3.3 Results on MIRFLICKR25K

We further evaluate NSH and competitive methods on MIRFLICKR25K, which contains less samples and labels (only 38) than NUS-WIDE. We adopt MIRFLICKR25K because it is more densely labeled, with each sample having 4.72 labels on average, and so we can use more strict similarities. In our experiments, we determine a pair as similar if they contain at least three common labels, making the task challenging in another aspect. Overall results are presented in table 5. Again, NSH attains the best performance with varying number of bits, demonstrating the effectiveness of the proposed method. More detailed results are given in figure 2, where we plot Hamming ranking precision of top-ranked samples under 16, 32 and 64 bits. KSH, CCA-ITQ, and SDH attain close results while NSH surpasses them by a notable margin.

4 Conclusion and future work

In this paper we propose a novel supervised hashing method *Natural Supervised Hashing* and show its superior performance on 4 image datasets. We attribute the success of NSH to the inner-product-preserving objective. We find that although labels are not optimal binary codes under commonly adopted evaluation criteria, it still gives impressive results on preserving semantic neighborhoods. Inspired by its good performance, we intend to formulate an objective function so that our learnt code can generate close inner product to label vectors. To enable more easy optimization and circumvent large $n \times n$ matrices, we manage to find a transformation which preserves inner products of each pair and so we can use it to bring close labels and our code without explicit computation on inner products. In the end, our objective takes two cases and both can be optimized in an alternating way. We optimize the objective with Procrustes solution and relaxation, leading to efficient learning process. In experiments, NSH consistently surpasses compared methods and attains best results in all 4 benchmark datasets, indicating the efficacy of the objective and learning method. In the end, we think that our method can still be improved, e.g. by utilizing more complex hash functions like neural networks. We intend to look in this direction in our future work.

5 Acknowledgements

This work is supported by NSFC (No. 61272247, 61533012, 61472075), the 863 National High Technology Research and Development Program of China (SS2015AA020501) and the Major Basic Research Program of Shanghai Science and Technology Committee (15JC1400103).

References

[Chua *et al.*, 2009] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national uni-

versity of singapore. In *Proceedings of the ACM international conference on image and video retrieval*, page 48. ACM, 2009.

[Datar *et al.*, 2004] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[Erin Liong *et al.*, 2015] Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou. Deep hashing for compact binary codes learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2475–2483, 2015.

[Ge *et al.*, 2014] Tiezheng Ge, Kaiming He, and Jian Sun. Graph cuts for supervised binary coding. In *Computer Vision–ECCV 2014*, pages 250–264. Springer, 2014.

[Gionis *et al.*, 1999] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[Gong and Lazebnik, 2011] Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 817–824. IEEE, 2011.

[Gong *et al.*, 2013] Yunchao Gong, Sudhakar Kumar, Henry Rowley, Svetlana Lazebnik, et al. Learning binary codes for high-dimensional data using bilinear projections. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 484–491. IEEE, 2013.

[Guillaumin *et al.*, 2010] Matthieu Guillaumin, Jakob Verbeek, and Cordelia Schmid. Multimodal semi-supervised learning for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 902–909. IEEE, 2010.

[Kong and Li, 2012] Weihao Kong and Wu-Jun Li. Isotropic hashing. In *Advances in Neural Information Processing Systems*, pages 1646–1654, 2012.

[Kulis and Darrell, 2009] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *Advances in neural information processing systems*, pages 1042–1050, 2009.

[Kulis and Grauman, 2009] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2130–2137. IEEE, 2009.

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

[Liu *et al.*, 2012] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2074–2081. IEEE, 2012.

[Norouzi and Blei, 2011] Mohammad Norouzi and David M Blei. Minimal loss hashing for compact binary codes. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 353–360, 2011.

[Shalev-Shwartz *et al.*, 2011] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[Shen *et al.*, 2015] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. Supervised discrete hashing. In *CVPR 2015: IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2015.

[Wang *et al.*, 2010] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Sequential projection learning for hashing with compact codes. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 1127–1134, 2010.

[Wang *et al.*, 2012] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for large-scale search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(12):2393–2406, 2012.

[Wang *et al.*, 2013] Jun Wang, Wei Liu, Andy X Sun, and Yu-Gang Jiang. Learning hash codes with listwise supervision. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 3032–3039. IEEE, 2013.

[Weiss *et al.*, 2009] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Advances in neural information processing systems*, pages 1753–1760, 2009.

[Xia *et al.*, 2014] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *Proceedings of the AAAI Conference on Artificial Intellignece*, pages 2156–2162, 2014.

[Xia *et al.*, 2015] Yan Xia, Kaiming He, Pushmeet Kohli, and Jian Sun. Sparse projections for high-dimensional binary codes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3332–3339, 2015.

[Xu *et al.*, 2013] Bin Xu, Jiajun Bu, Yue Lin, Chun Chen, Xiaofei He, and Deng Cai. Harmonious hashing. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1820–1826. AAAI Press, 2013.