# Supervised Hashing with Soft Constraints

Cong Leng, Jian Cheng, Jiaxiang Wu, Xi Zhang, Hanqing Lu
National Laboratory of Pattern Recognition
Institute of Automation, Chinese Academy of Sciences
Beijing, China
{cong.leng, jcheng, jiaxiang.wu, zhangxi, luhq}@nlpr.ia.ac.cn

## ABSTRACT

Due to the ability to preserve semantic similarity in Hamming space, supervised hashing has been extensively studied recently. Most existing approaches encourage two dissimilar samples to have maximum Hamming distance. This may lead to an unexpected consequence that two unnecessarily similar samples would have the same code if they are both dissimilar with another sample. Besides, in existing methods, all labeled pairs are treated with equal importance without considering the semantic gap, which is not conducive to thoroughly leverage the supervised information. We present a general framework for supervised hashing to address the above two limitations. We do not toughly require a dissimilar pair to have maximum Hamming distance. Instead, a soft constraint which can be viewed as a regularization to avoid over-fitting is utilized. Moreover, we impose different weights to different training pairs, and these weights can be automatically adjusted in the learning process. Experiments on two benchmarks show that the proposed method can easily outperform other state-of-the-art methods.

## Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval

## Keywords

Supervised Hashing; Soft Constraints; Weights; Boosting

## 1. INTRODUCTION

Hashing based approximate nearest neighbor (ANN) search methods have attracted much attention recently. Hashing methods map the two nearby points in the original space to close binary codes in a compact Hamming space. This enables very fast searching since Hamming distance can be efficiently calculated with XOR operation in modern CPU. According to whether supervised information is utilized or not in the training process, hashing methods can be divided into unsupervised and supervised categories. In the unsupervised setting, hashing methods such as Locality Sensitive Hashing (LSH) [1] and Iterative Quantization (ITQ) [2] attempt to preserve the data similarity defined in Euclidean space, e.g., $l_2$ distance. However, this is not sufficient for various practical applications such as image retrieval, where semantically similar neighbors are preferred.

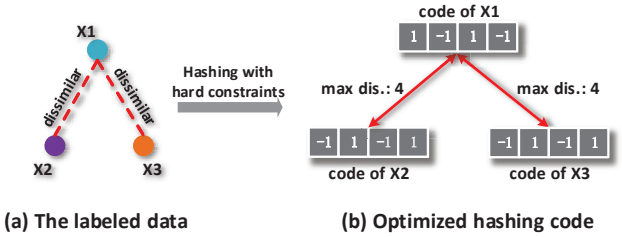



**Figure 1: Paradox in traditional supervised Hashing methods. Two unnecessarily similar data $x_2$ and $x_3$ will have the same code.**

In order to construct efficient hash functions that preserve the semantic similarity, supervised hashing methods [3, 6, 5, 4, 7] have been extensively studied. The supervised information here is typically based on some pairwise constraints, i.e., "A and B is similar" or "A and B is dissimilar", which is analogous to the "must link" and "cannot link" constraints in metric learning [8]. Some representative supervised hashing methods include Binary Reconstruction Embedding (BRE) [3], Kernel Supervised Hashing (KSH) [5] and Two Step Hashing (TSH) [4]. These supervised methods can be formally formulated with following objective [4]:

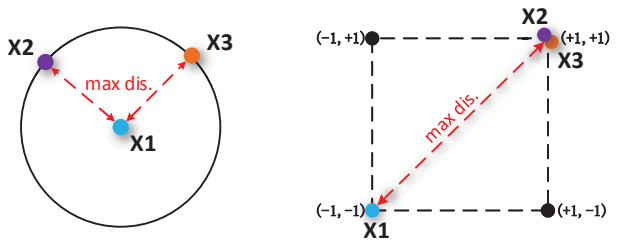
$$\min_{\Phi} \sum_{(x_i,x_j)\in\mathcal{L}} L\left(\Phi(x_i), \Phi(x_j); y_{ij}\right) \tag{1}$$

where $\Phi(x) \in \{-1, 1\}^r$ is the $r$ bits code of $x$. $L(\cdot)$ is a loss function that measures how well the codes match the ground truth $y_{ij}$. Different algorithms corresponds to different loss functions, for example, $l_2$ loss for BRE and KSH. Although promising performance has been shown from these methods, some limitations exist in them.

Inspired by metric learning, all these supervised methods attempt to learn codes whose Hamming distances are minimized on similar pairs and simultaneously maximized on dissimilar pairs. This principle is widely used in metric learning and proved to be effective. However, metric learning executes in continuous real number space while hashing executes in discrete Hamming space. Importantly, although it makes sense in metric learning, we argue that maximiz-


*CIKM'14,* November 3–7, 2014, Shanghai, China.

http://dx.doi.org/10.1145/2661829.2661937.




(a) metric learning in real number space

(b) binary codes learning in Hamming space

**Figure 2: Difference between real space and Hamming space. (a) The circle is the "boundary" of a continuous real space. Maximizing the distance to $x_1$, point $x_2$ and $x_3$ can be anywhere on this circle. (b) However, in $2d$ discrete Hamming space, $x_2$ and $x_3$ will collide in one corner.**

ing Hamming distance on dissimilar pairs, namely hard constraints, will lead to over-fitting in hashing. Fig.1 gives an illustration about our observation. In this example, point $x_1$ is labeled to be dissimilar with $x_2$ and $x_3$ separately, while the relationship between $x_2$ and $x_3$ is unknown. Under the hard constraints, both $x_2$ and $x_3$ would have optimized code that is completely opposite with $x_1$. As a result, $x_2$ and $x_3$ will have the same code. This is apparently unreasonable because $x_2$ and $x_3$ are not necessarily similar. This paradox derives from the difference between the continuous space and discrete space (Fig.2). In fact, all the supervised hashing methods with hard constraints contain an implicit assumption, namely, if both 'B' and 'C' are dissimilar with 'A', then 'B' and 'C' are similar. However, this assumption typically does not hold and leads to over-fitting.

In addition, existing methods treat all labeled pairs equally. If each labeled pair is taken as one constraint in hashing, some of them are easy to satisfy while some others not. This is because the gap between the feature space and semantic space. Sometimes two semantically similar points are also close in the feature space. These two points are easy to be embedded to similar codes even without supervised information. Meanwhile, other labeled pairs may be with larger semantic gap, and need more attention in the learning process. Therefore, treating different pairs with different importance is necessary.

In this paper, a general framework is presented to address the above two limitations. We propose to apply soft constraints to the dissimilar pairs. Specifically, instead of toughly requiring a dissimilar pair to have maximum Hamming distance in the objective, we just request them to be far enough in the Hamming space. This can be viewed as a regularization to avoid over-fitting in supervised hashing. Furthermore, we impose different weights to different training pairs, and these weights can be automatically adjusted with boosting technique in a batch-wise learning process. Experiments on two benchmarks show that the proposed method can significantly outperform other state-of-the-art supervised hashing methods.

# 2. THE PROPOSED APPROACH

First of all, some notations are defined as follows: Let $X = [x_1, x_2, \cdots, x_n]$ denote a set of $n$ data points, where $x_i \in \mathbb{R}^d$ is the $i$-th data point. For each $x_i$, its binary hashing code is denoted as $\Phi(x_i) = [h_1(x_i), h_2(x_i), \cdots, h_r(x_i)] \in \{-1, 1\}^r$, where $r$ is the code length and $\Phi = [h_1(\cdot), h_2(\cdot), \cdots, h_r(\cdot)]$ is a set of $r$ hash functions. $\mathcal{L}$ denotes the set of labeled data pairs. Two categories of label information, $\mathcal{S}$ and $\mathcal{D}$, are available. $(x_i, x_j) \in \mathcal{S}$ represents a similar-pair in which $x_i$ and $x_j$ are similar in semantic space, e.g., share the same labels. Similarly, $(x_i, x_j) \in \mathcal{D}$ is called a dissimilar-pair if two samples are far away in semantic space.

Without loss of generality, we establish our model based on Hamming affinity [5] in this paper. Specifically, Hamming affinity is defined as:

$$S_{ij} = \frac{\Phi(x_i)^T \Phi(x_j)}{r} \tag{2}$$

Obviously, $\Phi(x_i)^T \Phi(x_j) \in [-r, r]$ and $S_{ij} \in [-1, 1]$. In this case, the ground truth affinity in Eq.(1) is defined as:

$$y_{ij} = \begin{cases} 1, & \text{if } (x_i, x_j) \in \mathcal{S} \\ -1, & \text{if } (x_i, x_j) \in \mathcal{D} \end{cases}$$

Following [5], by adopting Euclidean loss function, the objective function in Eq.(1) can be written as:

$$\min_{\Phi} \sum_{(x_i, x_j) \in \mathcal{L}} \left( \frac{\Phi(x_i)^T \Phi(x_j)}{r} - y_{ij} \right)^2 \tag{3}$$

## 2.1 Soft Constraint and Weighted Loss

In Eq.(3) and the objectives of other supervised methods [3, 5], two dissimilar points are encouraged to have completely different codes so that the corresponding Hamming affinity will be -1 (i.e. maximal Hamming distance). This idea comes from metric learning and works very well in continuous real space [8]. However, as illustrated in Fig.1 and Fig.2, hashing executes in discrete Hamming space and the hard constraints will result in entirely opposite codes of a dissimilar pair, which brings over-fitting, e.g. two unnecessarily similar samples would have the same code.

With the target of hashing based ANN search, we only need to require the codes of a dissimilar pair to be far away enough but not necessarily entirely opposite. Considering an example that, if the difference between two data points in a pair $(x_i, x_j) \in \mathcal{D}$ is 90% of the total bits, their Hamming distance will be $0.9 \times r$ and the corresponding Hamming affinity will be -0.8, and this is typically enough to separate the two samples in Hamming space. That is to say, there is no need to restrict the Hamming affinity of a dissimilar pair to be -1. To this end, we modify the objective function in Eq.(3) to be:

$$\min_{\Phi} \sum_{(x_i, x_j) \in \mathcal{L}} \left( \frac{\Phi(x_i)^T \Phi(x_j)}{r} - \lambda_{ij} y_{ij} \right)^2 \tag{4}$$

and $\lambda_{ij}$ is defined as:

$$\lambda_{ij} = \begin{cases} 1, & \text{if } (x_i, x_j) \in \mathcal{S} \\ 0 < p < 1, & \text{if } (x_i, x_j) \in \mathcal{D} \end{cases} \tag{5}$$

where $p \in (0, 1)$ is a parameter in our method.

For any dissimilar pair $(x_i, x_j) \in \mathcal{D}$, $\lambda_{ij} y_{ij} = -p \in (-1, 0)$. In other words, we relax the original hard labels into a soft range for the dissimilar pairs. Hence, the optimized codes will have $\frac{1+p}{2} \times r$ bits to be different. In this paper, we call this kind of constraint as soft constraint. Obviously, under such soft constraints, the codes of $x_2$ and $x_3$ are not necessarily identical like in Fig.1. It is worth noting that for a similar

pair $(x_i, x_j) \in \mathcal{S}$, $\lambda_{ij} = 1$, we do not change the constraints on similar pairs, and keep to encourage similar samples in semantic space to have the same codes in Hamming space. Multiplying $y_{ij}$ with $\lambda_{ij}$ can be viewed as a kind of regularization to avoid over-fitting in supervised hashing. As we will find in the following experiments, this little change can greatly improve the quality of learned codes.

Eq.(4) treats each labeled pair as equally important, which fails to take the semantic gap into account. Regarding each labeled pair in $\mathcal{L}$ as a constraint, some of them are easy to satisfy while some others not. For the sake of maximally leveraging the supervised information, we need to treat different pairs with different importance. This can be implemented by giving a different weight $w_{ij}$ to each pair $(x_i, x_j) \in \mathcal{L}$, and we arrive at:

$$\min_{\Phi} \sum_{(x_i,x_j)\in\mathcal{L}} w_{ij} \left( \frac{\Phi(x_i)^T \Phi(x_j)}{r} - \lambda_{ij} y_{ij} \right)^2 \tag{6}$$

While it can be intuitively understood, an important question arises: how to determine the weight of each labeled pair? In the next section, we present a batch-wise learning approach to automatically determine the adaptive weights for different labeled pairs in the learning process.

## 2.2 Learn one batch at a time

Most of methods solve the optimized code by a single-shot optimization to the objective, i.e., learn all bits in a single run of the algorithm. However, for any a piece of binary code, it can be regarded as a concatenation of many pieces of shorter codes. As an example, one piece of 32-bits code can be considered as the result of concatenating two pieces of 16-bits codes, or four pieces of 8-bits codes. The Hamming affinity evaluated with the long code is the mean of those evaluated with the short ones. From this point of view, hashing can be understood as an ensemble learning process if we repeatedly generate short codes and then concatenate them. The weight of each labeled pair can be adjusted automatically with boosting trick in this process.

Specifically, in the first run, all labeled pairs are treated with equal importance, and we only generate a piece of short code $\Phi_1$ of $t$ bits (e.g. 4 bits) for each sample. In the second and following runs, the weight of each pair will be updated by considering the deviation of the Hamming affinity evaluated with the previous code to the ground truth. Higher weights will be imposed to the pairs with bigger deviation. In detail, in the $k$-th run, the weight can be defined as:

$$w_{ij}^{(k)} = \begin{cases} 1 - \frac{\Phi(x_i)^T \Phi(x_j)}{(k-1)\times t}, & \text{if} \quad (x_i, x_j) \in \mathcal{S} \\ max(\frac{\Phi(x_i)^T \Phi(x_j)}{(k-1)\times t} + p, 0), & \text{if} \quad (x_i, x_j) \in \mathcal{D} \end{cases} \tag{7}$$

where $\Phi(x) = [\Phi_1(x); \Phi_2(x); \cdots; \Phi_{k-1}(x)]$ is the $(k-1) \times t$ bits code of $x$ by concatenating the previous $k-1$ pieces of short codes $\Phi_i(x)|_{i=1}^{k-1}$.

Obviously, $\frac{\Phi(x_i)^T \Phi(x_j)}{(k-1)\times t}$ is the Hamming affinity evaluated with the previous code. For the similar pairs, $1 - \frac{\Phi(x_i)^T \Phi(x_j)}{(k-1)\times t}$ is the deviation to the ground truth affinity. The pairs with bigger deviation are associated with higher weights in the next iteration. Note that for the dissimilar pairs, a hinge-like function is used to measure the bias. This is to be consistent with the soft constraints we have used. For any dissimilar pair whose Hamming affinity is smaller than $-p$, the deviation is set to be zero. After $K$ iterations, we can get a piece of $K \times t$ bits code for each training data.

## 2.3 Optimization

A remaining problem is optimizing Eq.(6) to learn one piece of $t$ bits code for each training data. Here we follow the block coordinate descent (BCD) method which was also used in [5, 4]. In particular, BCD picks one bit to be optimized every time with other $t-1$ bits fixed. The optimization for the $m$-th bit can be written as:

$$\min_{\mathbf{z}_{(m)}\in\{-1,1\}^n} \sum_{(x_i,x_j)\in\mathcal{L}} w_{ij} l_m(z_{i,m}, z_{j,m}) \tag{8}$$

where $\mathbf{z}_{(m)}$ is the binary codes of the $m$-th bit. $l_m$ is the loss function defined on the $m$-th bit, i.e.

$$l_m(z_{i,m}, z_{j,m}) = L(z_{i,m}, z_{j,m}, \bar{\mathbf{z}}_i, \bar{\mathbf{z}}_j; \lambda_{ij} y_{ij}) \tag{9}$$

Here $L$ is the loss function defined in Eq.(4). $z_{i,m}$ is the binary code of the $i$-th sample and the $m$-th bit. $\bar{\mathbf{z}}_i$ is the binary codes of the $i$-th sample excluding the $m$-th bit.

When optimizing the $m$-th bit, as indicated in TSH [4], there are only two possible cases for the code of any pair, namely, same or different. We denote the loss of appointing the same code to $x_i$ and $x_j$ as $l_{m,i,j}^{(+)}$, and $l_{m,i,j}^{(-)}$ for appointing different code. By taking advantage of the Proposition 1 in [4], the optimization of Eq.(8) can be rewritten as:

$$\min_{\mathbf{z}_{(m)}\in\{-1,1\}^n} \sum_{(x_i,x_j)\in\mathcal{L}} w_{ij}(l_{m,i,j}^{(+)} - l_{m,i,j}^{(-)}) z_{i,m} z_{j,m} \tag{10}$$

Because $w_{ij}$, $l_{m,i,j}^{(+)}$ and $l_{m,i,j}^{(-)}$ are constants, the optimization can be written in a matrix form:

$$\min_{\mathbf{z}_{(m)}\in\{-1,1\}^n} \mathbf{z}_{(m)}^T \mathbf{A} \mathbf{z}_{(m)} \tag{11}$$

where the element of matrix $\mathbf{A}$ is $a_{ij} = w_{ij}(l_{m,i,j}^{(+)} - l_{m,i,j}^{(-)})$. The optimization of Eq. (11) have been well studied. To be specific, by dropping the binary constraints, the optimization becomes:

$$\min_{\mathbf{z}_{(m)}} \quad \mathbf{z}_{(m)}^T \mathbf{A} \mathbf{z}_{(m)} \quad s.t. \quad ||\mathbf{z}_{(m)}||_2^2 = n \tag{12}$$

The optimum solution is the eigenvector corresponding to the minimum eigenvalue of $\mathbf{A}$. Subsequently, the obtained solution will be quantized to $\{-1, 1\}^t$ with the sign function.

Until now, we have only optimized the binary codes for training data, which is not enough because hashing has to handle the out-of-sample extension problem, i.e., generating codes for new samples that are unseen before. Inspired by TSH [4], for every bit we regard the binary value $z_{i,m}$ as the pseudo label of training data $x_i$. Therefore the given training set has already been "labelled" by the above learning process, and we can learn a binary classifier $f^{(m)}$ based on it for every bit. The resulting binary classifiers $f^{(m)}|_{m=1}^{t}$ are taken as the hash function. In the experiments, we choose SVM with Gaussian kernel as classifier, which is widely used and of good performance.

# 3. EXPERIMENTS

We compare our method with several state-of-the-art approaches, including four supervised methods: BRE [3], MLH [6], KSH [5], TSH [4], and two representative unsupervised methods: LSH [1], ITQ [2]. Comparison experiments

**Table 1: Hamming ranking performance of different algorithms with different code lengthes on MNSIT and CIFAR-10. Mean Average Precision (MAP) is reported. The best results are highlighted in bold.**

| Methods | MNIST | | | | CIFAR-10 | | | |
|---|---|---|---|---|---|---|---|---|
| | 16-bits | 24-bits | 32-bits | 64-bits | 16-bits | 24-bits | 32-bits | 64-bits |
| LSH | 0.2343 | 0.2319 | 0.2536 | 0.3335 | 0.1282 | 0.1382 | 0.1398 | 0.1532 |
| ITQ | 0.4188 | 0.4375 | 0.4444 | 0.4629 | 0.1670 | 0.1707 | 0.1756 | 0.1798 |
| BRE | 0.5350 | 0.5592 | 0.5914 | 0.6010 | 0.1588 | 0.1637 | 0.1668 | 0.1760 |
| MLH | 0.7062 | 0.7433 | 0.7672 | 0.8011 | 0.1911 | 0.2072 | 0.2174 | 0.2462 |
| KSH | 0.7776 | 0.8008 | 0.8180 | 0.8268 | 0.2202 | 0.2379 | 0.2492 | 0.2683 |
| TSH | 0.5870 | 0.7978 | 0.8533 | 0.8793 | 0.2377 | 0.2685 | 0.2874 | 0.3117 |
| SCH_uw | 0.7314 | 0.8445 | 0.8650 | 0.8808 | 0.2666 | 0.2958 | 0.3102 | 0.3286 |
| SCH | **0.8532** | **0.8663** | **0.8733** | **0.8875** | **0.2885** | **0.3154** | **0.3193** | **0.3385** |

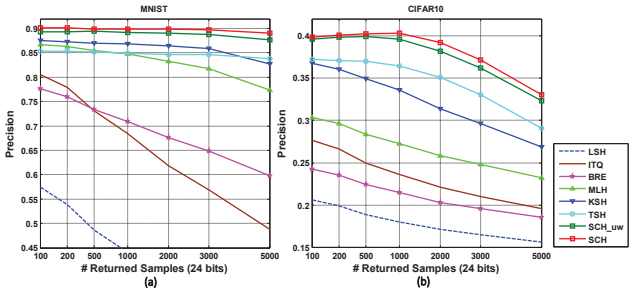



**Figure 3: Precision curves of different methods with 24 bits on MNIST and CIFAR-10**

are conducted on two widely used benchmarks: MNIST[1] and CIFAR-10[2]. MNIST consists of 784 dimensional 70,000 samples associated with digits from '0' to '9'. CIFAR-10 is a labeled subset of the Tiny Images dataset. 512 dimensional GIST descriptor is extracted to represent each image. For both datasets, we randomly select 1,000 samples to be the query set and the remainders as database. 1,000 samples in the database are used to randomly generate training pairs. Specifically, we suppose that for each sample in the training set, only the relationship to 500 other samples in this set are known. Thus, about 500,000 training pairs are available. We adopt the Hamming Ranking commonly used in the literature. All points in the database are ranked according to their Hamming distance to the query. The ground truth is defined as semantic neighbors based on label agreement.

To give a comprehensive validation of the proposed approach, we present two versions of our method. In the first version, denoted as SCH_uw, we only apply soft constraints to the learning process and ignore the weights of different pairs. The second version, denoted as SCH, considers both the soft constraints and weighted loss. In this version, the learning process optimizes 4 bits at each run and then adjusts the weight of each pair. We empirically set the parameter $p$ in Eq.(5) as $p = 0.6$.

**MAP scores:** The MAP scores of SCH, SCH_uw and other baselines are shown in Table 1. By leveraging side-information, the supervised methods like KSH and TSH can achieve significant improvement on the unsupervised methods like ITQ. The proposed SCH achieves the highest search accuracy on both two datasets. The optimization of our method is similar to that in TSH and KSH, but it is easy to find that SCH outperforms them with a large margin, especially on the CIFAR-10 dataset. More notably, even ignoring the weights of pairs, SCH_uw achieves the best results except SCH in most of settings. This confirms that the proposed soft constraints can effectively avoid over-fitting in the supervised hashing. By considering the weights of different pairs, SCH achieves further improvement on SCH_uw, which demonstrates that treating different pairs with different importance is beneficial to take full advantage of the supervised information.

**Precision Curves:** Fig.3 shows the precision curves of different methods with 24 bits on two datasets. Similar to the trends in Table 1, SCH works better than SCH_uw, which is the second best in all competitors. In Fig.3(a), the precision decreases in all hashing methods as the number of retrieved points increases, but our methods decrease more slowly and achieve a very high precision on MNIST even when 5,000 samples are returned. These results clearly show the superiority of the proposed methods over other state-of-the-art methods.

## 4. CONCLUSION

In this paper, we proposed a general framework for supervised hashing with soft constraints and weighted loss. Experiments on two benchmarks demonstrated the effectiveness of the proposed method.

## Acknowledgements

This work was supported in part by National Natural Science Foundation of China (Grant No. 61332016), 863 program (Grant No. 2014AA015104 and 2014AA015105).

## 5. REFERENCES

[1] M. Charikar. Similarity estimation techniques from rounding algorithm. In *STOC*, 2002.
[2] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011.
[3] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, 2009.
[4] G. Lin, C. Shen, D. Suter, and A. van den Hengel. A general two-step approach to learning-based hashing. In *ICCV*, 2013.
[5] W. Liu, J. Wang, R. Ji, Y. Jiang, and S. Chang. Supervised hashing with kernels. In *CVPR*, 2012.
[6] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *ICML*, 2011.
[7] M. Ou, P. Cui, F. Wang, J. Wang, W. Zhu, and S. Yang. Comparing apples to oranges: A scalable solution with heterogeneous hashing. In *KDD*, 2013.
[8] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning with application to clustering with side-information. In *NIPS*, 2003.

[1] `http://yann.lecun.com/exdb/mnist/`
[2] `http://www.cs.toronto.edu/~kriz/cifar.html`