LEARNING SUPERVISED BINARY HASHING: OPTIMIZATION VS DIVERSITY

Ramin Raziperchikolaei and Miguel Á. Carreira-Perpiñán

Electrical Engineering and Computer Science
University of California, Merced, USA

ABSTRACT

Binary hashing is a practical approach for fast, approximate retrieval in large image databases. The goal is to learn a hash function that maps high-dimensional images onto a binary vector such that Hamming distances approximate semantic similarities. The search is then fast by using hardware support for binary operations. Most hashing papers define a complicated objective function that couples the single-bit hash functions. A recent work has shown the surprising result that by learning the single-bit functions independently and making them diverse using ensemble learning techniques, one can achieve simpler optimization, faster training, and better retrieval results. In this paper, we study the interplay between optimization and diversity in learning good hash functions. We show that to achieve good hash functions, no matter how we optimize the objective, the diversity among the single-bit hash functions is a crucial element.

Index Terms— Image retrieval, ensemble learning, optimization

1. INTRODUCTION

In image retrieval problems, the goal is to search a large database of images and find the ones that are similar to the test images. This can be seen as a k-nearest-neighbors problem where we are interested in the nearest neighbors of the test images. We usually have a large database containing millions of images, where each image is represented by high-dimensional feature vectors. In this case, the exact search is impractical. Binary hashing is an efficient approach that solves this problem approximately, but very fast [1]. The idea is to learn a hash function that maps high-dimensional images into binary codes and then find the nearest neighbors in the binary space. This has two main advantages: (1) the search is fast because hardware operations can be used to compute the Hamming distances between the codes and (2) billions of images can be stored in the main memory of a single machine if the binary feature vectors are short enough.

The goal of binary hashing is to learn a b-bits hash function that preserves the similarities of the original images in the new binary space. There are two approaches to achieve this: optimization-based and ensemble-based approaches. In optimization-based approaches [2, 3, 4, 5, 6], all the single-bit hash functions are coupled in the objective function, using constraints or penalty terms. This leads to complicated objectives that are difficult to optimize and can not scale well. In the ensemble-based approach [7, 8], the objective function is defined over only a single-bit hash function. The b-bits hash function is learned by minimizing this objective b times independently and making the functions different using ensemble learning techniques. The ensemble-based approach has several advantages: simpler optimization problems, faster training, massive parallelism, scalability to large datasets, and better retrieval performance. We give a quick review of the ensemble-based approach in section 2.

This poses an intriguing question: *how important is diversity vs optimization in preserving neighborhoods and making the hash functions differ?* In section 3, we give a review of different ways to optimize a 1-bit objective function. In section 4, we investigate the importance of diversity by experimentally controlling the quality of the optimization of the 1-bit objective function with different algorithms and observing its effect on the learned b-bit hash function.

1.1. Related work

Data dependent hashing methods learn the hash function by minimizing an objective, defined over the training points. They perform better than data independent methods, which do not have any training (e.g. considering random hyperplanes [9] as the hash functions). In unsupervised hashing [10, 11, 12], the similarity between the points is defined based on their closeness in the Euclidean space. We focus on supervised hashing [3, 5] where the objective is to preserve the semantic similarity, where the points that are far away in Euclidean space can still be considered similar to each other.

Most hashing papers define a complicated objective function that couples the single-bit functions and then try to optimize it in an approximate way [2, 3, 13, 14, 5, 15, 6]. In [13, 14, 5], the objective is optimized over the binary codes in the first step and the hash functions are learned a posteriori. In [15, 6], one optimizes the objective function jointly over the codes and hash functions, while preserving the binary constraints. A recent method, *Independent Laplacian Hashing (ILH)* [7], proposes to learn the single-bit hash functions independently and make them diverse using ensemble diversity approaches. ILH can be improved by pruning a set of hash functions and selecting the training subsets locally [8].

2. INDEPENDENT LAPLACIAN HASHING (ILH)

Independent Laplacian Hashing (ILH) [7] is the first supervised hashing method that uses diversity techniques to make the hash functions differ. *The main idea of ILH is to optimize the same objective over a single-bit hash function b times independently, and make the single-bit functions different using techniques from the ensemble learning literature.* Specifically, assume that $\mathbf{h}(\cdot)$ is a hash function that takes an input $\mathbf{x} \in \mathbb{R}^D$ and outputs a single bit in $\{-1, +1\}$. ILH proposes to optimize the following objective b times independently:

$$\min_{\mathbf{h}} P(\mathbf{h}) = \mathbf{h}(\mathbf{X})\,\mathbf{A}\,\mathbf{h}(\mathbf{X})^T = \textstyle\sum_{n,m=1}^{N} a_{nm}\, h(\mathbf{x}_n)\, h(\mathbf{x}_m) \quad (1)$$

where $\mathbf{h}(\mathbf{X}) = (h(\mathbf{x}_1), \dots, h(\mathbf{x}_N)) \in \{-1, +1\}^N$ is a row vector of N bits, $\mathbf{A} = (a_{nm}) \in \mathbb{R}^{N \times N}$ is the affinity matrix for the training set $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N) \in \mathbb{R}^{D \times N}$, and the minimization is over the parameters of $\mathbf{h}$. For example, the affinity a_{mn} can be $+1$, -1 or 0 for similar, dissimilar or indifferent points $\mathbf{x}_n$ and $\mathbf{x}_m$. If the training sets differ across hash functions, then so do the affinity matrices and we learn different hash functions.

The ensemble-based approach proposed by ILH gives several advantages: (1) optimization problems are easier to solve, (2) the

Work supported by NSF award IIS–1423515.

retrieval results are better, (3) hash functions can be learned in parallel, and (4) it scales to large datasets. In [7], the best results were achieved when ILH used different training subsets to make the functions different. We consider this mechanism whenever we refer to ILH.

3. OPTIMIZING SINGLE HASH FUNCTIONS: REVIEW

In this section, we give a review of several representative optimization algorithms that have been used before in the hashing literature to optimize objective functions. We describe them for the case where we have a single hash function. With $b = 1$, they become particularly simple and efficient and it is instructive to understand this. This is important because all of these optimization algorithms can be used in ILH to minimizes $P(\mathbf{h})$ in eq. (1) approximately and learn each of the 1-bit hash functions. This also provides an apples-to-apples comparison between optimization algorithms on the same objective function, which is lacking in the literature of binary hashing, which has mostly focused on contributing new objective functions and/or hash functions and algorithms tailored to them. We compare these algorithms experimentally in section 4, to see: (1) which of these algorithms optimizes the objective function better, (2) whether a better optimization leads to a better retrieval, and (3) the critical role of diversity as we put the 1-bit hash functions together.

We focus on optimizing the objective function of eq. (1) over a single hash function h: $\mathbb{R}^D \rightarrow \{-1,+1\}$. Some of our development corresponds to the particular case when the hash function is linear: $h(\mathbf{x}) = \lrcorner(\mathbf{w}^T\mathbf{x})$, where $\lrcorner(t) = +1$ if $t \geq 0$ and -1 if $t < 0$, and we have augmented $\mathbf{x}$ with an extra constant element of value 1, so that the corresponding element in $\mathbf{w}$ represents a bias parameter. In this case the objective function is

$$\min_{\mathbf{w}\in\mathbb{R}^D} P(\mathbf{w}) = \lrcorner(\mathbf{w}^T\mathbf{X})\,\mathbf{A}\,\lrcorner(\mathbf{w}^T\mathbf{X})^T = \sum_{n,m=1}^N a_{nm}\,\lrcorner(\mathbf{w}^T\mathbf{x}_n)\,\lrcorner(\mathbf{w}^T\mathbf{x}_m). \quad (2)$$

We will describe several optimization algorithms. Two of them are applicable to any type of hash function (two-step and MAC) and the other two are only convenient for linear hash functions (direct alternating optimization and relaxed eigenproblem).

Before describing the algorithms, we present a basic characterization of the optimization problem to give an idea of its difficulty. Although this argument is valid for a more general parametric hash function, consider for simplicity the case of a linear hash function $h(\mathbf{x}) = \lrcorner(\mathbf{w}^T\mathbf{x})$. The objective $P(\mathbf{w})$ is piecewise constant over $\mathbf{w}$, because the number of possible values of $\mathbf{h}(\mathbf{X})$ is 2^N and thus the number of possible values of $P(\mathbf{w})$ is finite, and upper bounded by 2^N. The actual number of regions is much smaller than 2^N because some binary codes $\mathbf{z} \in \{-1,+1\}^N$ cannot be achieved by any linear function operating on $\mathbf{X}$. Also, if the matrix $\mathbf{A}$ has integer elements (0 or ±1 in many papers), the number of values of $P(\mathbf{w})$ is even smaller. Thus, if we vary $\mathbf{w} \in \mathbb{R}^D$ by a small amount, either $P(\mathbf{w})$ will not change (and stay within a piece of P of constant value), or it will change discontinuously (and move to another piece).

Let us look at the objective over just one element of $\mathbf{w}$, say w_d:

$$p_d(w_d) = \sum_{n,m=1}^N a_{nm}\,\lrcorner(w_d x_{dn} + c_{dn})\,\lrcorner(w_d x_{dm} + c_{dm}) \quad (3)$$

where $1 \leq d \leq D$, $c_{dn} = \sum_{i=1,i\neq d}^D w_i x_{in}$. The function $p_d(w_d)$ is piecewise constant in at most $N+1$ intervals $-\infty < -c_{d1}/x_{d1} \leq \cdots \leq -c_{dN}/x_{dN} < \infty$ (where we reassign w.l.o.g. the indices so that the values $-c_{dn}/x_{dn}$ are sorted increasingly, and consider only values for which $x_{dn} \neq 0$). The global minimum of $p_d(w_d)$ must occur at one (or more) of these intervals. These intervals can be represented by their midpoints (plus two arbitrary points in the infinite intervals at each end), and the global minimum can be found by evaluating $p_d(w_d)$ at those points.

Hence, the gradient of $P(\mathbf{w})$ is either 0 or it does not exist, so we cannot use gradient-based optimization, at least using the chain rule. The algorithms described below use other approaches.

3.1. Two-step optimization

Many binary hashing papers use this approach with a variety of objective functions and hash functions [16, 13, 10, 11, 14, 5]. The idea is to introduce the N-bit binary code $\mathbf{z} \in \{0,1\}^N$ and define the objective function over $\mathbf{z}$ instead of the hash functions $\mathbf{h}(\cdot)$ in eq. (1):

$$\min_{\mathbf{z}} E(\mathbf{z}) = \mathbf{z}\mathbf{A}\mathbf{z}^T \quad \text{with} \quad \mathbf{z} \in \{-1,+1\}^N. \quad (4)$$

First step is to optimize $E(\mathbf{z})$ over $\mathbf{z}$ regardless of the hash function, and the second step is to fit the hash function $\mathbf{h}$ to these codes.

As mentioned above, the minimization of $E(\mathbf{z})$ is an NP-complete problem, and there are various approximation algorithms that can be used. Here we capitalize on min-cut [17, 18], which has a well-developed implementation that can efficiently solve submodular $E(\mathbf{z})$ for large N. Following [5], we subdivide $\mathbf{A}$ into submodular groups of binary variables and run alternating optimization on these blocks, solving each with min-cut. A different algorithm described below that relaxes the binary codes into an eigenproblem and then fits the hash function is also a two-step approach.

3.2. MAC (method of auxiliary coordinates) optimization

The basic idea in the *method of auxiliary coordinates (MAC)* [19, 20] is to introduce auxiliary coordinates that break functional nesting and simplify the optimization. It has been applied to binary hashing with binary autoencoders [12] and affinity-based loss functions [6]. Following the latter, we write the "nested" problem (1) as an equivalent, constrained problem without nesting:

$$\min_{\mathbf{h},\mathbf{z}} E(\mathbf{z}) = \mathbf{z}\mathbf{A}\mathbf{z}^T \quad \text{s.t.} \quad \mathbf{z} = \mathbf{h}(\mathbf{X}) \in \{-1,+1\}^N. \quad (5)$$

We now solve the constrained problem using the quadratic-penalty method [21]. This involves solving a sequence of unconstrained problems for $\mu \rightarrow \infty$ of the form:

$$\min_{\mathbf{h},\mathbf{z}} E(\mathbf{z}) + \mu\,\|\mathbf{z} - \mathbf{h}(\mathbf{X})\|^2 = \mathbf{z}\mathbf{A}\mathbf{z}^T + \mu\,\|\mathbf{z} - \mathbf{h}(\mathbf{X})\|^2.$$

where $\mathbf{z} \in \{-1,+1\}^N$. Finally, we apply alternating optimization over the codes $\mathbf{z}$ and the hash function $\mathbf{h}$ in the following two steps:

- **z step (1D binary embedding):** optimize over $\mathbf{z}$ for fixed $\mathbf{h}$:

$$\min_{\mathbf{z}} \mathbf{z}\mathbf{A}\mathbf{z}^T + \mu\,\|\mathbf{z} - \mathbf{h}(\mathbf{X})\|^2 = \mathbf{z}\mathbf{A}\mathbf{z}^T - 2\mu\mathbf{h}(\mathbf{X})\mathbf{z}^T + \text{constant}$$

where we have used the fact that $z^2 = 1$ if $z \in \{-1,+1\}$. This is a binary optimization like that of $E(\mathbf{z})$ in eq. (4) except that it carries a "regularization" term $\mu\|\mathbf{z} - \mathbf{h}(\mathbf{X})\|^2$ that drives the codes $\mathbf{z}$ towards the function outputs $\mathbf{h}(\mathbf{X})$ with a weight μ that keeps increasing during training. This can be seen as a Markov random field with a unary potential $-2\mu\mathbf{h}(\mathbf{X})$ and a quadratic potential $E(\mathbf{z}) = \mathbf{z}\mathbf{A}\mathbf{z}^T$. We can optimize this with the same algorithms as for $E(\mathbf{z})$.

- **h step (fit classifier):** optimize over $\mathbf{h}$ for fixed $\mathbf{z}$:

$$\min_{\mathbf{h}} \|\mathbf{z} - \mathbf{h}(\mathbf{X})\|^2 = \sum_{n=1}^N (z_n - h(\mathbf{x}_n))^2 \quad (6)$$

$\mathbf{h}$ can be learned by fitting a classifier to the inputs $\mathbf{X}$ and labels $\mathbf{z}$.

MAC jointly optimizes over $\mathbf{z}$ and $\mathbf{h}$, while a two-step algorithm greedily optimizes first over $\mathbf{z}$ and then over $\mathbf{h}$, which is suboptimal. MAC improves over the two-step algorithm in binary autoencoders and affinity-based loss functions for binary hashing [12, 6].

3.3. Direct alternating optimization

Here, we minimize $P(\mathbf{w})$ by alternating optimization (coordinate descent) on $\mathbf{w} = (w_1, \dots, w_D)^T$, optimizing in turn $p_d(w_d)$ in eq. (3) over each element of $\mathbf{w}$. This was used in BRE [2] over all b bits, which noted the BRE objective was piecewise constant over a single entry of its $b \times D$ weight matrix. Our algorithm is for only one bit but for a more general objective (2), both of which result in a simpler and faster algorithm. The minimization over each w_d, $d = 1, \dots, D$ can be done exactly in polynomial time. As noted earlier, the objective function $p_d(w_d)$ over a single element $w_d \in \mathbb{R}$ of $\mathbf{w}$ has the form of a piecewise constant function on at most $N+1$ intervals of positive length, two of them infinite (at the ends), and the rest finite (the interior intervals). Therefore at least one of these intervals consists of global minimizers. Clearly, the global minimum of $p_d(w_d)$ can be found simply by evaluating p_d at one point in each interval. Naively, this would cost $\mathcal{O}((D + \kappa + \log N)N^2)$ for one element of $\mathbf{w}$ and therefore $\mathcal{O}((D + \kappa + \log N)DN^2)$ for all D elements, which constitutes one iteration of alternating optimization, where $1 \leq \kappa \leq N$ is the number of nonzeros per row of $\mathbf{A}$ (equal to the number of positive and negative neighbors for each point). Although polynomial in D and N, this is very slow, and would not scale beyond a few thousand points. It can be shown that the exact solution can be found in only $\mathcal{O}((\kappa + \log N)DN)$ time and $\mathcal{O}(N + D)$ space per iteration. The main idea is to compute both c_{dn} and $p_d(w_d)$ in eq. (3), exactly but fast incrementally (similar to the idea in [2]).

The alternating optimization terminates in a finite number of iterations because $p_d(w_d)$ can take 2^N different values at most, one for each possible code $\mathbf{z} = \mathbf{h}(\mathbf{X}) \in \{-1, +1\}^N$, and the algorithm only updates w_d when it decreases $p_d(w_d)$. In practice, it stops in very few iterations, typically no more than 10. Given its low cost per iteration, this makes it comparatively fast. The convergence point is very sensitive to the initial $\mathbf{w}$, and one should not initialize from $\mathbf{w} = \mathbf{0}$ (or small $\|\mathbf{w}\|$ in general).

Alternating optimization can also be applied to the objective function over the codes in eq. (4), one bit at a time, and this is the method of iterated conditional modes (ICM) [22]. ICM also converges in very few iterations and is sensitive to the initial codes. Methods based on graph cuts achieve better optima [23, 17, 18].

3.4. Relaxed eigenproblem and truncation

Relaxation is an approach that can be used directly or as initialization for a secondary optimization [16, 13, 10, 24, 3]. We consider two algorithms based on relaxation: one applied to the binary codes $\mathbf{z}$, and the other applied directly to the hash function weight vector $\mathbf{w}$.

- **Relaxing the binary codes $\mathbf{z}$** We apply a two-step approach but relax the step over the codes. That is, since $z_n^2 = 1$ for $n = 1, \dots, N$, then $\mathbf{z}\mathbf{z}^T = N$. Hence, relaxing eq. (4) we obtain:

$$\min_{\mathbf{z} \in \mathbb{R}^N} \mathbf{z}\mathbf{A}\mathbf{z}^T \quad \text{s.t.} \quad \mathbf{z}\mathbf{z}^T = 1. \tag{7}$$

This is a (typically) sparse eigenproblem of $N \times N$. Since in general $\mathbf{A}$ is not positive semidefinite, $\mathbf{z}$ is the eigenvector associated with the most negative eigenvalue. Having obtained $\mathbf{z}$, we binarize the codes $\mathbf{z}$ by thresholding their sign and train a classifier on them, as in the two-step approach above. We have used $\mathbf{z}\mathbf{z}^T = 1$ rather than $\mathbf{z}\mathbf{z}^T = N$ because it does not change the result of the binarization.

- **Relaxing directly on $\mathbf{w}$** Since $P(\mathbf{w})$ is invariant to scaling $\mathbf{w}$, we can set $\|\mathbf{w}\| = 1$. If we relax $P(\mathbf{w})$ by dropping the ⎍$(\cdot)$ step functions, we obtain the following eigenproblem:

$$\min_{\mathbf{w} \in \mathbb{R}^D} \mathbf{w}^T(\mathbf{X}\mathbf{A}\mathbf{X}^T)\mathbf{w} \quad \text{s.t.} \quad \mathbf{w}^T\mathbf{w} = 1. \tag{8}$$

This is a dense eigenproblem of $D \times D$. In general $\mathbf{X}\mathbf{A}\mathbf{X}^T$ is not positive semidefinite, so $\mathbf{w}$ is the eigenvector associated with the most negative eigenvalue.

Both of these relaxations are similar to spectral dimensionality reduction using a Laplacian objective and a latent space of dimension 1. Specifically, relaxing the binary codes in (7) is like solving a Laplacian eigenmaps problem [25] where the graph Laplacian matrix is $\mathbf{A}$, which is not positive semidefinite. Likewise, relaxing directly on $\mathbf{w}$ is similar to locality preserving projection (LPP) [26], which does:

$$\begin{aligned} &\min_{\mathbf{w}} \mathbf{y}\mathbf{A}\mathbf{y}^T \quad \text{s.t.} \quad \mathbf{y}\mathbf{y}^T = 1,\ \mathbf{y} = \mathbf{w}^T\mathbf{X} \\ &\iff \quad \min_{\mathbf{w}} \mathbf{w}^T(\mathbf{X}\mathbf{A}\mathbf{X}^T)\mathbf{w} \quad \text{s.t.} \quad \mathbf{w}^T(\mathbf{X}\mathbf{X}^T)\mathbf{w} = 1 \end{aligned} \tag{9}$$

so it solves a generalized eigenproblem.

4. EXPERIMENTS

We investigate the interplay of optimization and diversity in learning good hash functions. We use CIFAR [27] dataset that contains 60 000 images in 10 classes with 58 000/2 000 images as the training/test sets. Each image is represented by $D = 320$ SIFT features [28]. The positive neighbors and ground-truth set of a point $\mathbf{x}$ are all the points with the same label as the label of $\mathbf{x}$. The retrieved set contains the r nearest neighbors of the query in the Hamming space.

Most hashing papers use a small subset of training set to train the hash functions [2, 3, 14, 15]. In our experiments, all the hash functions are trained using training subsets of 5 000 points that are selected randomly from the dataset. To evaluate the methods, we search the entire dataset to find the nearest neighbors of a query. We consider linear SVMs (trained with LIBLINEAR [29]) as the hash function for the cut, relaxed-codes and MAC methods.

4.1. Optimization vs diversity in learning hash functions

We first compare the six optimization algorithms that are explained in section 3: Two-step optimization (cut), MAC optimization, direct alternating optimization (alt opt), relaxing the binary codes in eq. (7) (relaxed-codes), relaxing the weights $\mathbf{w}$ in eq. (8) (relaxed-weights), relaxing the weights $\mathbf{w}$ in eq. (9) similar to the locally linear projection (relaxed-LPP). After that, we show the importance of diversity as we combine the hash functions and compare the role of diversity and optimization in learning the hash functions.

Does a better 1-bit optimization lead to a better retrieval? We first create 64 training subsets of 5000 points, randomly from the CIFAR dataset. Then, for each of the 1-bit optimization algorithms, we optimize the objective function of eq. (2) on one of the subsets to learn a 1-bit hash function. Repeating this for all the subsets gives us 64 1-bit hash functions for each of the algorithms. In the left panel of fig. 1, we plot the distribution of the objective function error (bottom) and the distribution of the precision (top) of the 1-bit hash functions for each of the algorithms. To be able to compare the two plots easily, we compute the distribution of $-P(\mathbf{h})$ instead of $P(\mathbf{h})$.

By looking at the left panel of fig. 1, we find that *by decreasing the value of the objective function (better optimization), we achieve a better precision.* We can see this in different ways. Hash functions of relaxed-LPP give the lowest objective function error and also the highest precision. Also, relaxed-codes gives the worst optimization and 1-bit precision values. MAC always beats cut by decreasing the objective error of each of the hash functions and improving the 1-bit precisions. We can also compare the variance in the value of the objective function with the variance in the 1-bit precision. Methods like relaxed-weights and relaxed-LPP have small variance in the distribution of their objective function values and precisions. The other four methods have a large variance in both of the distributions.

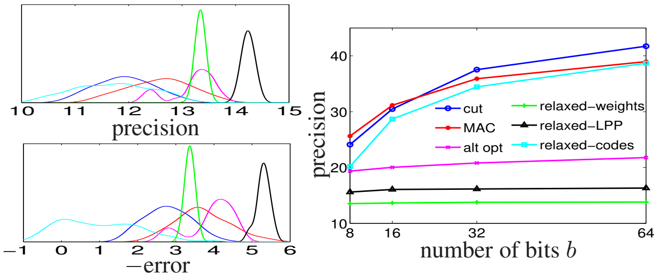

Fig. 1. Optimization vs diversity in the 1-bit case. *Left plots*: distribution of the 1-bit precision (top) and negative error (bottom, defined as $-P(\mathbf{h}) = -\mathbf{h}(\mathbf{X})\mathbf{A}\mathbf{h}(\mathbf{X})^T \times 10^5$) over 64 1-bit functions. The diversity of the hash functions corresponds to the variance of the distributions. Lower error correlates with higher precision. *Right plot*: b-bit precision of the hashing methods. Methods with low diversity perform poorly even if they use 1-bit functions with higher precision.

Optimization vs diversity. We put the 1-bit hash functions together, create groups of 8 to 64 bits and report their precisions in the right panel of fig. 1. It illustrates the importance of the diversity in combining the hash functions. While the 1-bit hash functions of the relaxed-LPP give the best precisions, the combination of those bits does not perform better than the 1-bit case and gives one of the worst precisions comparing to other methods. This could happen when the 64 hash functions (and consequently the binary codes) are very similar and we gain almost nothing by putting them together. *For a b-bit hash function, a better* 1*-bit optimization only improves the precision significantly if there is sufficient diversity.*

By looking at the distribution of the objective error and precision of a specific method, we can judge its diversity. As the variance of the error increases, the variance of the precision also increases, which leads to more diversity and better precision by combining the hash functions. Methods like cut, MAC, and relaxed-codes, which have a high variance in their precision and error, show much better performance as we combine the 1-bit hash functions.

Does the hash function fit the binary codes accurately? We explore this by repeating the previous experiment and comparing binary codes and hash functions of the two-step methods. In fig. 2, we report the distribution of the objective error for the 1-bit binary codes and hash functions using eq. (4) and eq. (2), respectively. cut-codes (relaxed-codes) denotes the binary codes achieved by the cut (relaxed-codes) optimization and cut-lin (relaxed-lin) and cut-ker (relaxed-ker) denote the linear and kernel SVMs learned based on the codes. The figure shows that the cut algorithm finds much better local optima than the relaxed-codes. It also shows that *there is a large gap between the output of the hash functions and binary codes.* We can make the gap a little smaller by using kernel SVMs or other nonlinear hash functions.

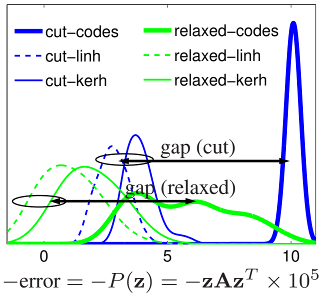

Fig. 2. Gap between the desired binary codes and the codes produced by the learned hash functions. We plot the distribution of the 1-bit errors in both cases, for the two-step methods (cut and relaxed). For the 1-bit hash functions we set $\mathbf{z} = \mathbf{h}(\mathbf{X})$ and compute the errors.

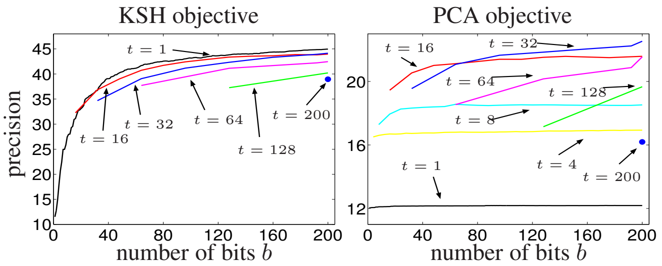

Fig. 3. Optimization vs diversity in the t-bit case. ILH learns b/t independent t-bit hash functions for KSH and PCA.

Optimization vs diversity in the t-bit case. We create b/t training subsets where each of them containing 5 000 points from CIFAR dataset. We train t-bit hash functions (instead of the 1-bit ones) on each of the subsets to learn the final b-bit hash function ($b = 200$ in this experiment). To learn the t-bit functions, we use the KSH and PCA objective functions. In general, both objective functions couple the 1-bit hash functions to make them different.

The parameter t determines the source of the diversity among the hash functions. For $t = 1$, we optimize 200 1-bit objective functions, each of them on a different subset. In this case, independent training sets are the source of diversity. For $t = 200$, we optimize the coupled objective function on only one subset of points, and use the optimization technique to make the functions diverse.

Fig. 3 (left) shows the results of using the KSH objective function. The best results are achieved when we use independent training sets to make the functions diverse ($t = 1$). This is because optimizing the single bit objective is easier, the functions are diverse and trained on larger sets of points (implicitly). As we decrease the value of t (use more optimization than diversity), the precision decreases.

Consider the PCA objective (bagged PCA [30]) in fig. 3 (right). Here, the precision is minimal for $t = 1$, because in PCA the first few principal components (weights of the hash functions) of different subsets of a dataset are very similar to each other. The reason is that the direction of the variance in these subsets is very similar to the direction of the variance in the main dataset. By increasing t, we increase the number of principal components from each subset, which makes the hash functions diverse with better performance. When we use $t > 32$, we couple the hash functions more and more, use more optimization (lose diversity), and we achieve worse precisions.

We conclude that independent training sets do not necessarily lead to diverse hash functions. *We may need both optimization technique and independent sets to make the hash functions diverse.* Also, while both optimization and diversity are important, diversity is crucial to get significant improvement as we use groups of bits.

5. CONCLUSION

To learn a supervised b-bit hash function, most hashing papers try to approximately optimize a complicated objective function that couples all the single-bit hash functions. A recent paper proposed a new approach (ILH): define an objective function over a single-bit hash function, optimize it independently b times but make the B single-bit functions differ using ensemble-based diversity approaches.

We have investigated the interplay between diversity and optimization in learning good hash functions by studying several 1-bit optimization algorithms. Among our findings, a result that emerges consistently is that, although improving the optimization can improve the results of hash functions, diversity is the absolutely crucial element, without which hashing becomes ineffective. This explains the success of simple methods that construct bits independently, either with learning (like ILH) or without it (like LSH).

6. REFERENCES

[1] Kristen Grauman and Rob Fergus, "Learning binary hash codes for large-scale image search," in *Machine Learning for Computer Vision*, 2013.

[2] Brian Kulis and Trevor Darrell, "Learning to hash with binary reconstructive embeddings," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[3] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang, "Supervised hashing with kernels," in *Proc. of the 2012 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'12)*, 2012.

[4] Mohammad Norouzi and David Fleet, "Minimal loss hashing for compact binary codes," in *Proc. of the 28th Int. Conf. Machine Learning (ICML 2011)*, 2011.

[5] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton van den Hengel, and David Suter, "Fast supervised hashing with decision trees for high-dimensional data," in *Proc. of the 2014 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'14)*, 2014.

[6] Ramin Raziperchikolaei and Miguel Á. Carreira-Perpiñán, "Optimizing affinity-based binary hashing using auxiliary coordinates," in *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[7] Miguel Á. Carreira-Perpiñán and Ramin Raziperchikolaei, "An ensemble diversity approach to supervised binary hashing," in *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[8] Ramin Raziperchikolaei and Miguel Á. Carreira-Perpiñán, "Learning independent, diverse binary hash functions: Pruning and locality," in *Proc. of the 17th IEEE Int. Conf. Data Mining (ICDM 2016)*, 2016.

[9] Alexandr Andoni and Piotr Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Comm. ACM*, 2008.

[10] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang, "Hashing with graphs," in *Proc. of the 28th Int. Conf. Machine Learning (ICML 2011)*, 2011.

[11] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin, "Iterative quantization: A Procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2013.

[12] Miguel Á. Carreira-Perpiñán and Ramin Raziperchikolaei, "Hashing with binary autoencoders," in *Proc. of the 2015 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'15)*, 2015.

[13] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu, "Self-taught hashing for fast similarity search," in *Proc. of the 33rd ACM Conf. Research and Development in Information Retrieval (SIGIR 2010)*,2010.

[14] Guosheng Lin, Chunhua Shen, David Suter, and Anton van den Hengel, "A general two-step approach to learning-based hashing," in *Proc. 14th Int. Conf. Computer Vision (ICCV'13)*, 2013.

[15] Tiezheng Ge, Kaiming He, and Jian Sun, "Graph cuts for supervised binary coding," in *Proc. 13th European Conf. Computer Vision (ECCV'14)*, 2014.

[16] Yair Weiss, Antonio Torralba, and Rob Fergus, "Spectral hashing," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[17] Yuri Boykov, Olga Veksler, and Ramin Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2001.

[18] Vladimir Kolmogorov and Ramin Zabih, "What energy functions can be minimized via graph cuts?," *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2003.

[19] Miguel Á. Carreira-Perpiñán and Weiran Wang, "Distributed optimization of deeply nested systems," arXiv:1212.5921 [cs.LG], 2012.

[20] Miguel Á. Carreira-Perpiñán and Weiran Wang, "Distributed optimization of deeply nested systems," in *Proc. of the 17th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2014)*, 2014.

[21] Jorge Nocedal and Stephen J. Wright, *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering, 2006.

[22] Julian Besag, "On the statistical-analysis of dirty pictures," *J. Statistical Society B*, 1986.

[23] D. M. Greig, B. T. Porteous, and A. H. Seheult, "Exact maximum a posteriori estimation for binary images," *J. Statistical Society B*, 1989.

[24] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang, "Semi-supervised hashing for large scale search," *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2012.

[25] Mikhail Belkin and Partha Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, 2003.

[26] Xiaofei He and Partha Niyogi, "Locality preserving projections," in *Advances in Neural Information Processing Systems (NIPS)*, 2004.

[27] Alex Krizhevsky, "Learning multiple layers of features from tiny images," M.S. thesis, Dept. of Computer Science, University of Toronto, 2009.

[28] Aude Oliva and Antonio Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. J. Computer Vision*, 2001.

[29] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin, "LIBLINEAR: A library for large linear classification," *J. Machine Learning Research*, 2008.

[30] Cong Leng, Jian Cheng, Ting Yuan, Xiao Bai, and Hanqing Lu, "Learning binary codes with bagging PCA," in *Proc. of the 25th European Conf. Machine Learning (ECML–14)*, 2014.