Hashing with Binary Autoencoders

Miguel Á. Carreira-Perpiñán   Ramin Raziperchikolaei
Electrical Engineering and Computer Science, University of California, Merced
`http://eecs.ucmerced.edu`

January 4, 2015

Abstract

An attractive approach for fast search in image databases is binary hashing, where each high-dimensional, real-valued image is mapped onto a low-dimensional, binary vector and the search is done in this binary space. Finding the optimal hash function is difficult because it involves binary constraints, and most approaches approximate the optimization by relaxing the constraints and then binarizing the result. Here, we focus on the binary autoencoder model, which seeks to reconstruct an image from the binary code produced by the hash function. We show that the optimization can be simplified with the method of auxiliary coordinates. This reformulates the optimization as alternating two easier steps: one that learns the encoder and decoder separately, and one that optimizes the code for each image. Image retrieval experiments, using precision/recall and a measure of code utilization, show the resulting hash function outperforms or is competitive with state-of-the-art methods for binary hashing.

1 Introduction

We consider the problem of binary hashing, where given a high-dimensional vector $\mathbf{x} \in \mathbb{R}^D$, we want to map it to an L-bit vector $\mathbf{z} = \mathbf{h}(\mathbf{x}) \in \{0,1\}^L$ using a hash function $\mathbf{h}$, while preserving the neighbors of $\mathbf{x}$ in the binary space. Binary hashing has emerged in recent years as an effective technique for fast search on image (and other) databases. While the search in the original space would cost $\mathcal{O}(ND)$ in both time and space, using floating point operations, the search in the binary space costs $\mathcal{O}(NL)$ where $L \ll D$ and the constant factor is much smaller. This is because the hardware can compute binary operations very efficiently and the entire dataset (NL bits) can fit in the main memory of a workstation. And while the search in the binary space will produce some false positives and negatives, one can retrieve a larger set of neighbors and then verify these with the ground-truth distance, while still being efficient.

Many different hashing approaches have been proposed in the last few years. They formulate an objective function of the hash function $\mathbf{h}$ or of the binary codes that tries to capture some notion of neighborhood preservation. Most of these approaches have two things in common: $\mathbf{h}$ typically performs dimensionality reduction ($L < D$) and, as noted, it outputs binary codes ($\mathbf{h}\colon \mathbb{R}^D \rightarrow \{0,1\}^L$). The latter implies a step function or binarization applied to a real-valued function of the input $\mathbf{x}$. Optimizing this is difficult. In practice, most approaches follow a two-step procedure: first they learn a real hash function ignoring the binary constraints and then the output of the resulting hash function is binarized (e.g. by thresholding or with an optimal rotation). For example, one can run a continuous dimensionality reduction algorithm (by optimizing its objective function) such as PCA and then apply a step function. This procedure can be seen as a "filter" approach (Kohavi and John, 1998) and is suboptimal: in the example, the thresholded PCA projection is not necessarily the best thresholded linear projection (i.e., the one that minimizes the objective function under all thresholded linear projections). To obtain the latter, we must optimize the objective jointly over linear mappings and thresholds, respecting the binary constraints while learning $\mathbf{h}$; this is a "wrapper" approach (Kohavi and John, 1998). In other words, optimizing real codes and then projecting them onto the binary space is not the same as optimizing the codes in the binary space.

In this paper we show that this joint optimization, respecting the binary constraints during training, can actually be carried out reasonably efficiently. The idea is to use the recently proposed *method of auxiliary coordinates (MAC)* (Carreira-Perpiñán and Wang, 2012, 2014). This is a general strategy to transform an original problem involving a nested function into separate problems without nesting, each of which can be solved more easily. In our case, this allows us to reduce drastically the complexity due to the binary constraints. We focus on *binary autoencoders*, i.e., where the code layer is binary. We believe we are the first to apply MAC to this model and construct an efficient optimization algorithm. Section 3 describes the binary autoencoder model and objective function. Section 4 derives a training algorithm using MAC and explains how, with carefully implemented steps, the optimization in the binary space can be carried out efficiently, and parallelizes well. Our hypothesis is that constraining the optimization to the binary space results in better hash functions and we test this in experiments (section 6), using several performance measures: the traditional precision/recall, as well as the reconstruction error and an entropy-based measure of code utilization (which we propose in section 5). These show that linear hash functions resulting from optimizing a binary autoencoder using MAC are consistently competitive with the state-of-the-art, even when the latter uses nonlinear hash functions or more sophisticated objective functions for hashing.

2 Related work

The most basic hashing approaches are data-independent, such as Locality-Sensitive Hashing (LSH) (Andoni and Indyk, 2008), which is based on random projections and thresholding, and kernelized LSH (Kulis and Grauman, 2012). Generally, they are outperformed by data-dependent methods, which learn a specific hash function for a given dataset in an unsupervised or supervised way. We focus here on unsupervised, data-dependent approaches. These are typically based on defining an objective function (usually based on dimensionality reduction) either of the hash function or the binary codes, and optimizing it. However, this is usually achieved by relaxing the binary codes to a continuous space and thresholding the resulting continuous solution. For example, spectral hashing (Weiss et al., 2009) is essentially a version of Laplacian eigenmaps where the binary constraints are relaxed and approximate eigenfunctions are computed that are then thresholded to provide binary codes. Variations of this include using AnchorGraphs (Liu et al., 2011) to define the eigenfunctions, or obtaining the hash function directly as a binary classifier using the codes from spectral hashing as labels (Zhang et al., 2010). Other approaches optimize instead a nonlinear embedding objective that depends on a continuous, parametric hash function, which is then thresholded to define a binary hash function (Torralba et al., 2008; Salakhutdinov and Hinton, 2009); or an objective that depends on a thresholded hash function, but the threshold is relaxed during the optimization (Norouzi and Fleet, 2011). Some recent work has tried to respect the binary nature of the codes or thresholds by using alternating optimization directly on an objective function, over one entry or one row of the weight matrix in the hash function (Kulis and Darrell, 2009; Neyshabur et al., 2013), or over a subset of the binary codes (Lin et al., 2013, 2014). Since the objective function involves a large number of terms and all the binary codes or weights are coupled, the optimization is very slow. Also, Lin et al. (2013, 2014) learn the hash function after the codes have been fixed, which is suboptimal.

The closest model to our binary autoencoder is Iterative Quantization (ITQ) (Gong et al., 2013), a fast and competitive hashing method. ITQ first obtains continuous low-dimensional codes by applying PCA to the data and then seeks a rotation that makes the codes as close as possible to binary. The latter is based on the optimal discretization algorithm of Yu and Shi (2003), which finds a rotation of the continuous eigenvectors of a graph Laplacian that makes them as close as possible to a discrete solution, as a postprocessing for spectral clustering. The ITQ objective function is

$$\min_{\mathbf{B},\mathbf{R}} \|\mathbf{B} - \mathbf{V}\mathbf{R}\|^2 \qquad \text{s.t.} \qquad \mathbf{B} \in \{-1,+1\}^{NL},\ \mathbf{R}\ \text{ orthogonal} \tag{1}$$

where $\mathbf{V}$ of $N \times L$ are the continuous codes obtained by PCA. This is an NP-complete problem, and a local minimum is found using alternating optimization over $\mathbf{B}$, with solution $\mathbf{B} = \operatorname{sgn}(\mathbf{VR})$ elementwise, and over $\mathbf{R}$, which is a Procrustes alignment problem with a closed-form solution based on a SVD. The final hash function is $\mathbf{h}(\mathbf{x}) = \operatorname{sgn}(\mathbf{Wx})$, which has the form of a thresholded linear projection. Hence, ITQ is a postprocessing of the PCA codes, and it can be seen as a suboptimal approach to optimizing a binary autoencoder, where the binary constraints are relaxed during the optimization (resulting in PCA), and then one "projects" the continuous codes back to the binary space. Semantic hashing (Salakhutdinov and Hinton, 2009) also uses an autoencoder objective with a deep encoder (consisting of stacked RBMs), but again its optimization uses heuristics that are not guaranteed to converge to a local optimum: either training it as a continuous problem with backpropagation and then applying a threshold to the encoder (Salakhutdinov and Hinton, 2009), or rounding the encoder output to 0 or 1 during the backpropagation forward pass but ignoring the rounding during the backward pass (Krizhevsky and Hinton, 2011).

3 Our hashing models: binary autoencoder and binary factor analysis

We consider a well-known model for continuous dimensionality reduction, the (continuous) autoencoder, defined in a broad sense as the composition of an encoder $\mathbf{h}(\mathbf{x})$ which maps a real vector $\mathbf{x} \in \mathbb{R}^D$ onto a real code vector $\mathbf{z} \in \mathbb{R}^L$ (with $L < D$), and a decoder $\mathbf{f}(\mathbf{z})$ which maps $\mathbf{z}$ back to $\mathbb{R}^D$ in an effort to reconstruct $\mathbf{x}$. Although our ideas apply more generally to other encoders, decoders and objective functions, in this paper we mostly focus on the least-squares error with a linear encoder and decoder. As is well known, the optimal solution is PCA.

For hashing, the encoder maps continuous inputs onto *binary* code vectors with L bits, $\mathbf{z} \in \{0,1\}^L$. Let us write $\mathbf{h}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x})$ ($\mathbf{W}$ includes a bias by having an extra dimension $x_0 = 1$ for each $\mathbf{x}$) where $\mathbf{W} \in \mathbb{R}^{L\times(D+1)}$ and $\sigma(t)$ is a step function applied elementwise, i.e., $\sigma(t) = 1$ if $t \geq 0$ and $\sigma(t) = 0$ otherwise (we can fix the threshold at 0 because the bias acts as a threshold for each bit). Our *desired hash function* will be $\mathbf{h}$, and it should minimize the following problem, given a dataset of high-dimensional patterns $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$:

$$E_{\text{BA}}(\mathbf{h},\mathbf{f}) = \sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{f}(\mathbf{h}(\mathbf{x}_n))\|^2 \tag{2}$$

which is the usual least-squares error but where the code layer is binary. Optimizing this nonsmooth function is difficult and NP-complete. Where the gradients do exist wrt $\mathbf{W}$ they are zero nearly everywhere. We call this a *binary autoencoder (BA)*.

We will also consider a related model (see later):

$$E_{\text{BFA}}(\mathbf{Z},\mathbf{f}) = \sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\|^2 \qquad \text{s.t.} \qquad \mathbf{z}_n \in \{0,1\}^L,\ n = 1,\dots,N \tag{3}$$

where $\mathbf{f}$ is linear and we optimize over the decoder $\mathbf{f}$ and the binary codes $\mathbf{Z} = (\mathbf{z}_1, \dots, \mathbf{z}_N)$ of each input pattern. Without the binary constraint, i.e., $\mathbf{Z} \in \mathbb{R}^{L\times N}$, this model dates back to the 50s and is sometimes called least-squares factor analysis (Whittle, 1952), and its solution is PCA. With the binary constraints, the problem is NP-complete because it includes as particular case solving a linear system over $\{0,1\}^L$, which is an integer LP feasibility problem. We call this model (least-squares) *binary factor analysis (BFA)*. We believe this model has not been studied before, at least in hashing. A hash function $\mathbf{h}$ can be obtained from BFA by fitting a binary classifier of the inputs to each of the L code bits. It is a filter approach, while the BA is the optimal (wrapper) approach, since it optimizes jointly over $\mathbf{f}$ and $\mathbf{h}$.

Let us compare BFA (without bias term) with ITQ in eq. (1). BFA takes the form $\min_{\mathbf{Z},\mathbf{A}} \|\mathbf{X} - \mathbf{A}\mathbf{Z}\|^2$ s.t. $\mathbf{Z} \in \{0,1\}^{NL}$. The main difference is that in BFA the binary variables do not appear in separable form and so the step over them cannot be solved easily.

4 Optimization of BA and BFA using the method of auxiliary coordinates (MAC)

We use the recently proposed *method of auxiliary coordinates (MAC)* (Carreira-Perpiñán and Wang, 2012, 2014). The idea is to break nested functional relationships judiciously by introducing variables as equality constraints. These are then solved by optimizing a penalized function using alternating optimization over the original parameters and the coordinates, which results in a coordination-minimization (CM) algorithm. Recall eq. (2), this is our nested problem, where the model is $\mathbf{y} = \mathbf{f}(\mathbf{h}(\mathbf{x}))$. We introduce as auxiliary coordinates the outputs of $\mathbf{h}$, i.e., the codes for each of the N input patterns, and obtain the following equality-constrained problem:

$$\min_{\mathbf{h},\mathbf{f},\mathbf{Z}} \sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\|^2 \qquad \text{s.t.} \qquad \mathbf{z}_n = \mathbf{h}(\mathbf{x}_n) \in \{0,1\}^L,\ n = 1,\dots,N. \tag{4}$$

```
input $\mathbf{X}_{D\times N} = (\mathbf{x}_1,\dots,\mathbf{x}_N)$, $L \in \mathbb{N}$
Initialize $\mathbf{Z}_{L\times N} = (\mathbf{z}_1,\dots,\mathbf{z}_N) \in \{0,1\}^{LN}$
for $\mu = 0 < \mu_1 < \dots < \mu_\infty$
  for $l = 1,\dots,L$                                                h step
    $h_l \leftarrow$ fit SVM to $(\mathbf{X}, \mathbf{Z}_{\cdot l})$
  $\mathbf{f} \leftarrow$ least-squares fit to $(\mathbf{Z}, \mathbf{X})$                              f step
  for $n = 1,\dots,N$                                                Z step
    $\mathbf{z}_n \leftarrow \arg\min_{\mathbf{z}_n \in \{0,1\}^L} \|\mathbf{y}_n - \mathbf{f}(\mathbf{z}_n)\|^2 + \mu \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2$
  if $\mathbf{Z} = \mathbf{h}(\mathbf{X})$ then stop
return $\mathbf{h}$, $\mathbf{Z} = \mathbf{h}(\mathbf{X})$
```

Figure 1: Binary autoencoder MAC algorithm.

Note the codes are binary. We now apply the quadratic-penalty method (it is also possible to apply the augmented Lagrangian method instead; Nocedal and Wright, 2006) and minimize the following objective function while progressively increasing μ, so the constraints are eventually satisfied:

$$E_Q(\mathbf{h},\mathbf{f},\mathbf{Z};\mu) = \sum_{n=1}^{N} \left(\|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\|^2 + \mu \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2 \right) \qquad \text{s.t.} \qquad \mathbf{z}_n \in \{0,1\}^L,\ n = 1,\dots,N. \quad (5)$$

Now we apply alternating optimization over $\mathbf{Z}$ and $(\mathbf{h},\mathbf{f})$. This results in the following two steps:

- Over $\mathbf{Z}$ for fixed $(\mathbf{h},\mathbf{f})$, the problem separates for each of the N codes. The optimal code vector for pattern $\mathbf{x}_n$ tries to be close to the prediction $\mathbf{h}(\mathbf{x}_n)$ while reconstructing $\mathbf{x}_n$ well.
- Over $(\mathbf{h},\mathbf{f})$ for fixed $\mathbf{Z}$, we obtain $L+1$ independent problems for each of the L single-bit hash functions (which try to predict $\mathbf{Z}$ optimally from $\mathbf{X}$), and for $\mathbf{f}$ (which tries to reconstruct $\mathbf{X}$ optimally from $\mathbf{Z}$).

We can now see the advantage of the auxiliary coordinates: the individual steps are (reasonably) easy to solve (although some work is still needed, particularly for the $\mathbf{Z}$ step), and besides they exhibit significant parallelism. We describe the steps in detail below. The resulting algorithm alternates steps over the encoder (L classifications) and decoder (one regression) and over the codes (N binary proximal operators; Rockafellar, 1976; Combettes and Pesquet, 2011). During the iterations, we allow the encoder and decoder to be mismatched, since the encoder output does not equal the decoder input, but they are coordinated by $\mathbf{Z}$ and as μ increases the mismatch is reduced. The overall MAC algorithm to optimize a BA is in fig. 1.

Although a MAC algorithm can be shown to produce convergent algorithms as $\mu \to \infty$ with a differentiable objective function, we cannot apply the theorem of Carreira-Perpiñán and Wang (2012) because of the binary nature of the problem. Instead, we show that our algorithm converges to a local minimum for a *finite* μ, where "local minimum" is understood as in k-means: a point where $\mathbf{Z}$ is globally minimum given $(\mathbf{h},\mathbf{f})$ and vice versa. The following theorem is valid for any choice of $\mathbf{h}$ and $\mathbf{f}$, not just linear.

Theorem 4.1. *Assume the steps over* $\mathbf{h}$ *and* $\mathbf{f}$ *are solved exactly. Then the MAC algorithm for the binary autoencoder stops at a finite* μ.

Proof. μ appears only in the $\mathbf{Z}$ step, and if $\mathbf{Z}$ does not change there, $\mathbf{f}$ and $\mathbf{h}$ will not change either, since the $\mathbf{h}$ and $\mathbf{f}$ steps are exact. The $\mathbf{Z}$ step over $\mathbf{z}_n$ minimizes $\|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\|^2 + \mu\|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2$, and from theorem 4.3 we have that $\mathbf{h}(\mathbf{x}_n)$ is a global minimizer if $\mu > \|\mathbf{x}_n - \mathbf{f}(\mathbf{h}(\mathbf{x}_n))\|^2$. The statement follows from the fact that $\|\mathbf{x}_n - \mathbf{f}(\mathbf{z})\|^2$ is bounded over all n, $\mathbf{z}$ and $\mathbf{f}$. Let us prove this fact. Clearly this holds for fixed $\mathbf{f}$ because $(n, \mathbf{z}_n)$ take values on a finite set, namely $\{1,\dots,N\} \times \{0,\dots,2^L - 1\}$. As for $\mathbf{f}$, even if the set of functions $\mathbf{f}$ is infinite, the number of different functions $\mathbf{f}$ that are possible is finite, because $\mathbf{f}$ results from an exact fit to $(\mathbf{Z},\mathbf{X})$, where $\mathbf{X}$ is fixed and the set of possible $\mathbf{Z}$ is finite (since each z_{nl} is binary). □

The minimizers of $E_Q(\mathbf{h},\mathbf{f},\mathbf{Z};\mu)$ trace a path as a function of $\mu \geq 0$ in the $(\mathbf{h},\mathbf{f},\mathbf{Z})$ space. BA and BFA can be seen as the limiting cases of $E_Q(\mathbf{h},\mathbf{f},\mathbf{Z};\mu)$ when $\mu \to \infty$ and $\mu \to 0^+$, respectively (for BFA, $\mathbf{f}$

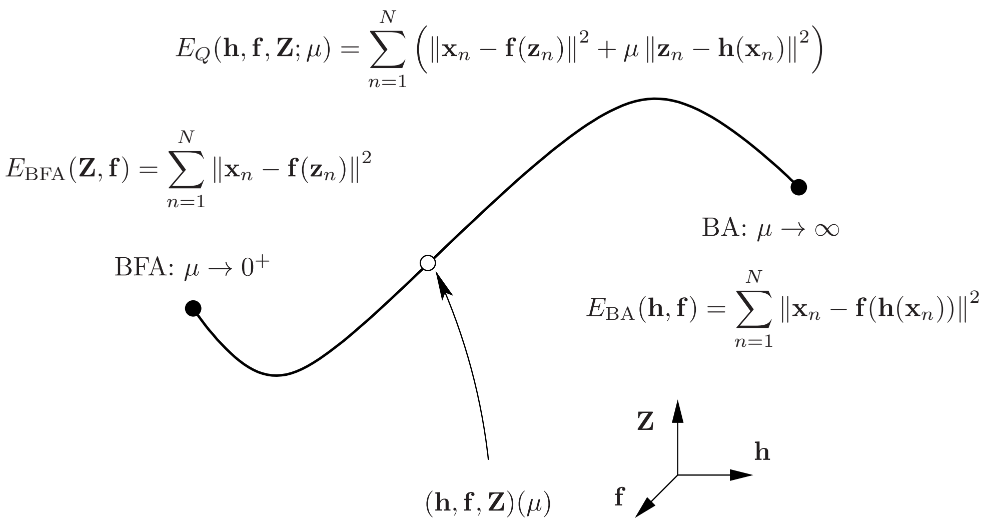

$E_Q(\mathbf{h},\mathbf{f},\mathbf{Z};\mu) = \sum_{n=1}^N \left(\|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\|^2 + \mu \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2 \right)$
$E_{\text{BFA}}(\mathbf{Z},\mathbf{f}) = \sum_{n=1}^N \|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\|^2$
BA: $\mu \to \infty$
BFA: $\mu \to 0^+$
$E_{\text{BA}}(\mathbf{h},\mathbf{f}) = \sum_{n=1}^N \|\mathbf{x}_n - \mathbf{f}(\mathbf{h}(\mathbf{x}_n))\|^2$
$\mathbf{Z}$
$\mathbf{h}$
$\mathbf{f}$
$(\mathbf{h},\mathbf{f},\mathbf{Z})(\mu)$

Figure 2: A continuous path induced by the quadratic-penalty objective function as a function of μ from BFA to BA. The objective functions of BA, BFA and the quadratic-penalty objective are related as shown in fig. 2. The minimizers of the quadratic-penalty objective $E_Q(\mathbf{h},\mathbf{f},\mathbf{Z};\mu)$ trace a path $(\mathbf{h},\mathbf{f},\mathbf{Z})(\mu)$ continuously as a function of $\mu \geq 0$ in the $(\mathbf{h},\mathbf{f},\mathbf{Z})$ space. The BFA and BA objective functions correspond to the limits $\mu \to 0^+$ and $\mu \to \infty$, respectively.

and $\mathbf{Z}$ can be optimized independently from $\mathbf{h}$, but $\mathbf{h}$ must optimally fit the resulting $\mathbf{Z}$). Figure 2 shows graphically the connection between the BA and BFA objective functions, as the two ends of the continuous path in $(\mathbf{h},\mathbf{f},\mathbf{Z})$ space in which the quadratic-penalty function $E_Q(\mathbf{h},\mathbf{f},\mathbf{Z};\mu)$ is defined.

In practice, to learn the BFA model we set μ to a small value and keep it constant while running the BA algorithm. As for BA itself, we increase μ (times a constant factor, e.g. 2) and iterate the $\mathbf{Z}$ and $(\mathbf{h},\mathbf{f})$ steps for each μ value. Usually the algorithm stops in 10 to 15 iterations, when no further changes to the parameters occur.

4.1 f step

With a linear decoder this is a simple linear regression

$$\min_{\mathbf{A},\mathbf{b}} \sum_{n=1}^N \|\mathbf{x}_n - \mathbf{A}\mathbf{z}_n - \mathbf{b}\|^2$$

with data $(\mathbf{Z},\mathbf{X})$. A bias parameter $\mathbf{b}$ is necessary, for example to be able to use $\{-1,+1\}$ instead of $\{0,1\}$ equivalently as bit values. The solution of this regression is (ignoring the bias for simplicity) $\mathbf{A} = \mathbf{X}\mathbf{Z}^T(\mathbf{Z}\mathbf{Z}^T)^{-1}$ and can be computed in $\mathcal{O}(NDL)$. Note the constant factor in the $\mathcal{O}$-notation is small because $\mathbf{Z}$ is binary, e.g. $\mathbf{X}\mathbf{Z}^T$ involves only sums, not multiplications.

4.2 h step

This has the following form

$$\min_{\mathbf{h}} \sum_{n=1}^N \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2 = \min_{\mathbf{W}} \sum_{n=1}^N \|\mathbf{z}_n - \sigma(\mathbf{W}\mathbf{x}_n)\|^2 = \sum_{l=1}^L \min_{\mathbf{w}_l} \sum_{n=1}^N (z_{nl} - \sigma(\mathbf{w}_l^T\mathbf{x}_n))^2. \tag{6}$$

Since $\mathbf{Z}$ and $\sigma(\cdot)$ are binary, $\|\cdot\|^2$ is the Hamming distance and the objective function is the number of misclassified patterns, so it separates for each bit. So it is a classification problem for each bit, using as labels the auxiliary coordinates, where h_l is a linear classifier (a perceptron). However, rather than minimizing this, we will solve an easier, closely related problem: fit a linear SVM h_l to $(\mathbf{X}, \mathbf{Z}_{\cdot l})$ where we use a high penalty for misclassified patterns but optimize the margin plus the slack. Besides being easier (by reusing well-developed codes for SVMs), this surrogate loss has the advantage of making the solution unique (no local optima) and generalizing better to test data (maximum margin). Also, although we used a quadratic penalty, the spirit of penalty methods is to penalize constraint violations $(\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n))$ increasingly. Since in the limit $\mu \to \infty$ the constraints are satisfied exactly, the classification error using $\mathbf{h}$ is zero, hence the linear SVM will find an optimum of the nested problem anyway. We use LIBLINEAR (Fan et al., 2008) with warm start (i.e., the SVM optimization is initialized from the previous iteration's SVM). Note the L SVMs and the decoder function $\mathbf{f}$ can be trained in parallel.

4.3 Z step

From eq. (5), this is a binary optimization on NL variables, but it separates into N independent optimizations each on only L variables, with the form of a binary proximal operator (Moreau, 1962; Rockafellar, 1976; Combettes and Pesquet, 2011) (where we omit the index n):

$$\min_{\mathbf{z}} e(\mathbf{z}) = \|\mathbf{x} - \mathbf{f}(\mathbf{z})\|^2 + \mu\|\mathbf{z} - \mathbf{h}(\mathbf{x})\|^2 \quad \text{s.t.} \quad \mathbf{z} \in \{0,1\}^L.$$

Thus, although the problem over each $\mathbf{z}_n$ is binary and NP-complete, a good or even exact solution may be obtained, because practical values of L are small (typically 8 to 32 bits). Further, because of the intensive computation and large number of independent problems, this step can take much advantage of parallel processing.

We have spent significant effort into making this step efficient while yielding good, if not exact, solutions. Before proceeding, let us show[1] how to reduce the problem, which as stated uses a matrix of $D \times L$, to an equivalent problem using a matrix of $L \times L$.

Theorem 4.2. *Let $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{A} \in \mathbb{R}^{D \times L}$, with QR factorisation $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q}$ is of $D \times L$ with $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ and $\mathbf{R}$ is upper triangular of $L \times L$, and $\mathbf{y} = \mathbf{Q}^T\mathbf{x} \in \mathbb{R}^L$. The following two problems have the same minima over $\mathbf{z}$:*

$$\min_{\mathbf{z} \in \{0,1\}^L} \|\mathbf{x} - \mathbf{Az}\|^2 \quad \min_{\mathbf{z} \in \{0,1\}^L} \|\mathbf{y} - \mathbf{Rz}\|^2. \tag{7}$$

Proof. Let $(\mathbf{Q}\ \mathbf{Q}_\perp)$ of $D \times D$ be orthogonal, where the columns of $\mathbf{Q}_\perp$ are an orthonormal basis of the nullspace of $\mathbf{Q}^T$. Then, since orthogonal matrices preserve Euclidean distances, we have: $\|\mathbf{x} - \mathbf{Az}\|^2 = \left\|(\mathbf{Q}\ \mathbf{Q}_\perp)^T(\mathbf{x} - \mathbf{Az})\right\|^2 = \left\|\mathbf{Q}^T(\mathbf{x} - \mathbf{Az})\right\|^2 + \left\|\mathbf{Q}_\perp^T(\mathbf{x} - \mathbf{Az})\right\|^2 = \|\mathbf{y} - \mathbf{Rz}\|^2 + \left\|\mathbf{Q}_\perp^T\mathbf{x}\right\|^2$, where the term $\left\|\mathbf{Q}_\perp^T\mathbf{x}\right\|^2$ does not depend on $\mathbf{z}$. □

This achieves a speedup of $2D/L$ (where the 2 factor comes from the fact that the new matrix is triangular), e.g. this is 40× if using 16 bits with $D = 320$ GIST features in our experiments. Henceforth, we redefine the $\mathbf{z}$ step as $\min e(\mathbf{z}) = \|\mathbf{y} - \mathbf{Rz}\|^2 + \mu\|\mathbf{z} - \mathbf{h}(\mathbf{x})\|^2$ s.t. $\mathbf{z} \in \{0,1\}^L$.

Enumeration For small L, this can be solved *exactly* by enumeration, at a worst-case runtime cost $\mathcal{O}(L^2 2^L)$, but with small constant factors in practice (see accelerations below). $L = 16$ is perfectly practical in a workstation without parallel processing for the datasets in our experiments.

Alternating optimization For larger L, we use alternating optimization over groups of g bits (where the optimization over a g-bit group is done by enumeration and uses the same accelerations). This converges to a local minimum of the $\mathbf{Z}$ step, although we find in our experiments that it finds near-global optima *if using a good initialization.* Intuitively, it makes sense to warm-start this, i.e., to initialize $\mathbf{z}$ to the code found in the previous iteration's $\mathbf{Z}$ step, since this should be close to the new optimum as we converge. However, empirically we find that the codes change a lot in the first few iterations, and that the following initialization works better (in leading to a lower objective value) in early iterations: we solve the relaxed problem on $\mathbf{z}$ s.t. $\mathbf{z} \in [0,1]^L$ rather than $\{0,1\}^L$. This is a strongly convex bound-constrained quadratic program (QP) in L variables for $\mu > 0$ and its unique minimizer can be found efficiently.

[1]This result can also be derived by expanding the norms into an $L \times L$ matrix $\mathbf{A}^T\mathbf{A}$ and computing its Cholesky decomposition. However, the product $\mathbf{A}^T\mathbf{A}$ squares the singular values of $\mathbf{A}$ and loses precision because of roundoff error (Golub and van Loan, 1996, p. 237ff; Nocedal and Wright, 2006, pp. 251).

We can further speed up the solution by noting that we have N QPs with some common, special structure. The objective is the sum of a term having the same matrix $\mathbf{R}$ for all QPs, and a term that is separable in $\mathbf{z}$. We have developed an ADMM algorithm (Carreira-Perpiñán, 2014) that is very simple, parallelizes or vectorizes very well, and reuses matrix factorizations over all N QPs. It is 10–100× faster than Matlab's `quadprog`. We warm-start it from the continuous solution of the QP in the previous $\mathbf{Z}$ step.

In order to binarize the continuous minimizer for $\mathbf{z}_n$ we could simply round its L elements, but instead we apply a greedy procedure that is efficient and better (though still suboptimal). We optimally binarize from bit 1 to bit L by evaluating the objective function for bit l in $\{0,1\}$ with all remaining elements fixed (elements 1 to $l-1$ are already binary and $l+1$ to L are still continuous) and picking the best. Essentially, this is one pass of alternating optimization but having continuous values for some of the bits.

Finally, we pick the best of the binarized relaxed solution or the warm-start value and run alternating optimization. This ensures that the quadratic-penalty function (5) decreases monotonically at each iteration.

Accelerations Naively, the enumeration involves evaluating $e(\mathbf{z})$ for 2^L (or 2^g) vectors, where evaluating $e(\mathbf{z})$ for one $\mathbf{z}$ costs on average roughly $L+1$ multiplications and $\frac{1}{4}L^2$ sums. This enumeration can be sped up or pruned while still finding a global minimum by using upper bounds on e, incremental computation of e, and necessary and sufficient conditions for the solution. Essentially, we need not evaluate every code vector, or every bit of every code vectors; we know the solution will be "near" $\mathbf{h}(\mathbf{x})$; and we can recognize the solution when we find it.

Call $\mathbf{z}^*$ a global minimizer of $e(\mathbf{z})$. An initial, good upper bound is $e(\mathbf{h}(\mathbf{x})) = \left\|\mathbf{y} - \mathbf{R}\,\mathbf{h}(\mathbf{x})\right\|^2$. In fact, we have the following sufficient condition for $\mathbf{h}(\mathbf{x})$ to be a global minimizer. (We give it generally for any decoder $\mathbf{f}$.)

Theorem 4.3. *Let $e(\mathbf{z}) = \left\|\mathbf{x} - \mathbf{f}(\mathbf{z})\right\|^2 + \mu\left\|\mathbf{z} - \mathbf{h}(\mathbf{x})\right\|^2$. Then: (1) A global minimizer $\mathbf{z}^*$ of $e(\mathbf{z})$ is at a Hamming distance from $\mathbf{h}(\mathbf{x})$ of $\frac{1}{\mu}\left\|\mathbf{x} - \mathbf{f}(\mathbf{h}(\mathbf{x}))\right\|^2$ or less. (2) If $\mu > \left\|\mathbf{x} - \mathbf{f}(\mathbf{h}(\mathbf{x}))\right\|^2$ then $\mathbf{h}(\mathbf{x})$ is a global minimizer.*

Proof. $e(\mathbf{z}^*) = \left\|\mathbf{x} - \mathbf{f}(\mathbf{z}^*)\right\|^2 + \mu\left\|\mathbf{z}^* - \mathbf{h}(\mathbf{x})\right\|^2 \leq e(\mathbf{h}(\mathbf{x})) = \left\|\mathbf{x} - \mathbf{f}(\mathbf{h}(\mathbf{x}))\right\|^2 \Rightarrow \left\|\mathbf{z}^* - \mathbf{h}(\mathbf{x})\right\|^2 \leq \frac{1}{\mu}\left\|\mathbf{x} - \mathbf{f}(\mathbf{h}(\mathbf{x}))\right\|^2$. (2) follows because the Hamming distance is integer. $\square$

As μ increases and $\mathbf{h}$ improves, this bound becomes more effective and more of the N patterns are pruned. Upon convergence, the $\mathbf{Z}$ step costs only $\mathcal{O}(NL^2)$. If we do have to search for a given $\mathbf{z}_n$, we keep a running bound (current best minimum) $\bar{e} = \left\|\mathbf{y}_n - \mathbf{R}\bar{\mathbf{z}}\right\|^2 + \mu\left\|\bar{\mathbf{z}} - \mathbf{h}(\mathbf{x}_n)\right\|^2$, and we scan codes in increasing Hamming distance to $\mathbf{h}(\mathbf{x}_n)$ up to a distance of $\bar{e}/\mu$. Thus, we try first the codes that are more likely to be optimal, and keep refining the bound as we find better codes.

Second, since $\left\|\mathbf{y} - \mathbf{R}\mathbf{z}\right\|^2$ separates over dimensions $1,\dots,L$, we evaluate it incrementally (dimension by dimension) and stop as soon as we exceed the running bound.

Finally, there exist global optimality necessary and sufficient conditions for binary quadratic problems that are easy to evaluate (Beck and Teboulle, 2000; Jeyakumar et al., 2007) (see appendix A). This allows us to recognize the solution as soon as we reach it and stop the search (rather than do a linear search of all values, keeping track of the minimum). These conditions can also determine whether the continuous solution to the relaxed QP is a global minimizer of the binary QP.

4.4 Schedule for the penalty parameter μ

The only user parameters in our method are the initialization for the binary codes $\mathbf{Z}$ and the schedule for the penalty parameter μ (sequence of values $0 < \mu_1 < \cdots < \infty$), since we use a penalty or augmented Lagrangian method. In general with these methods, setting the schedule requires some tuning in practice. Fortunately, this is simplified in our case for two reasons. 1) We need not drive $\mu \to \infty$ because *termination occurs at a finite μ and can be easily detected*: whenever $\mathbf{Z} = \mathbf{h}(\mathbf{X})$ at the end of the $\mathbf{Z}$ step, no further changes to the parameters can occur. This gives a practical stopping criterion. 2) In order to generalize well to unseen data, we stop iterating not when we (sufficiently) optimize $E_Q(\mathbf{h}, \mathbf{f}, \mathbf{Z}; \mu)$, but *when the precision in a validation set decreases*. This is a form of early stopping that guarantees that we improve (or leave unchanged) the initial $\mathbf{Z}$, and besides is faster. The initialization for $\mathbf{Z}$ and further details about the schedule for μ appear in section 6.

5 Measuring code utilization using entropy: effective number of bits

Here we propose an evaluation measure of binary hash functions that has not been used before as far as we know. Any binary hash function maps a population of high-dimensional real vectors onto a population of L-bit vectors (where L is fixed). Intuitively, a good hash function should make best use of the available codes and use each bit equally (since no bit is preferable to any other bit). For example, if we have $L = 32$ bits and $N = 10^6$ distinct real vectors, a good hash function would ideally assign a different 32-bit code to each vector, in order to avoid collisions. Given an L-bit hash function $\mathbf{h}$ and a dataset $\mathbf{X}$ of N real vectors $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^D$, we then obtain the N L-bit binary codes $\mathbf{z}_1, \dots, \mathbf{z}_N \in \{0,1\}^L$. We can measure the *code utilization* of the hash function $\mathbf{h}$ for the dataset $\mathbf{X}$ by the *entropy of the code distribution*, $S(P(\mathbf{h}(\mathbf{X})))$, defined as follows. Let c_i be the number of vectors that map to binary code i for $i = 0, \dots, 2^L - 1$. Then the (sample) code distribution P is a discrete probability distribution defined over the L-bit integers $0, \dots, 2^L - 1$ and has probability $p_i = \frac{c_i}{N}$ for code i, since $p_0 + \cdots + p_{2^L-1} = N$, i.e., the code probabilities are the normalized counts computed over the N data points. This works whether N is smaller or larger than 2^L (if $N < 2^L$ there will be unused codes). If the dataset is a sample of a distribution of high-dimensional vectors, then P is an estimate of the code usage induced by the hash function $\mathbf{h}$ for the distribution P based on a sample of size N. The entropy of P is

$$S(P) = -\sum_{i=0}^{2^L-1} p_i \log_2 p_i, \tag{8}$$

which is measured in bits. The entropy S is a real number and satisfies $S \in [0, \min(L, \log_2 N)]$. It is 0 when all N codes are coincident (hence only one of the 2^L available codes are used). It is L when $N \geq 2^L$ (the number of available codes is no more than the number of data points) and $p_i = 2^{-L}$ for all i (uniform distribution). It is $\log_2 N$ when $N < 2^L$ and all N codes are distinct. Hence, the entropy is large the more available codes are used, and the more uniform their use is. Since the entropy is measured in bits and cannot be more than L, S can be said to measure the *effective number of bits* L_{eff} of the code distribution induced by the hash function $\mathbf{h}$ on dataset $\mathbf{X}$.

A good hash function will have a good code utilization, making use of the available codes to avoid collisions and preserve neighbors. However, it is crucial to realize that an optimal code utilization does by itself not necessarily result in an optimal hash function, and it is not a fully reliable proxy for precision/recall. Indeed, code utilization is not directly related to the distances between data vectors, and it is easy to construct hash functions that produce an optimal code utilization but are not necessarily very good in preserving neighbors. For example, we can pick the first hash function as a hyperplane that splits the space into two half spaces each with half the data points (this is a high-dimensional median). Then, we do this within each half to set the second hash function, etc. This generates a binary decision tree with oblique cuts, which is impractical because it has 2^L hyperplanes, one per internal node. If the real vectors follow a Gaussian distribution (or any other axis-symmetric distribution) in $\mathbb{R}^D$, then using as hash functions any L thresholded principal components will give maximum entropy (as long as $L \leq D$). This is because each of the L hash functions is a hyperplane that splits the space into two half spaces containing half of the vectors, and the hyperplanes are orthogonal. This gives the thresholded PCA (tPCA) method mentioned earlier, which is generally not competitive with other methods, as seen in our experiments. More generally, we may expect tPCA and random projections through the mean to achieve high code utilization, because for most high-dimensional datasets (of arbitrary distribution), most low-dimensional projections are approximately Gaussian (Diaconis and Freedman, 1984).

With this caveat in mind, code utilization measured in effective number of bits L_{eff} is still a useful evaluation measure for hash functions. It also has an important advantage: it does not depend on any user parameters. In particular, it does not depend on the ground truth size (K nearest neighbors in data space) or retrieved set size (given by the k nearest neighbors or the vectors with codes within Hamming distance r). This allows us to compare all binary hashing methods with a single number L_{eff} (for a given number of bits L). We report values of L_{eff} in the experiments section and show that it indeed correlates well with precision in terms of the ranking of methods, particularly for methods that use the same model and objective function (such as the binary autoencoder reconstruction error with ITQ and MAC).

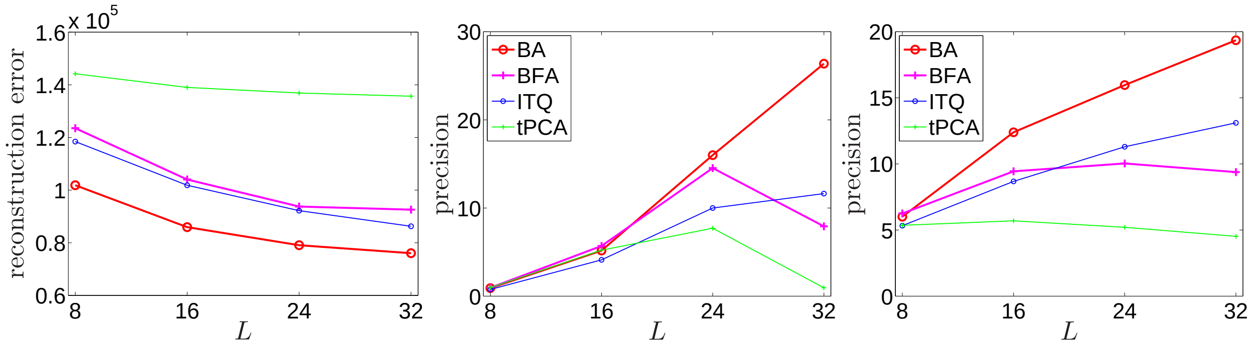


Figure 3: Wrapper vs filter optimization: BA objective function (*left*) and precision using neighbors within Hamming distance $r = 2$ (*middle*) and using $k = 50$ nearest neighbors (*right*).

6 Experiments

We used three datasets in our experiments, commonly used as benchmarks for image retrieval. (1) CIFAR (Krizhevsky, 2009) contains 60 000 32×32 color images in 10 classes. We ignore the labels in this paper and use $N = 50\,000$ images as training set and 10 000 images as test set. We extract $D = 320$ GIST features (Oliva and Torralba, 2001) from each image. (2) NUS-WIDE (Chua et al., 2009) contains $N = 269\,648$ high-resolution color images and use $N = 161\,789$ for training and 107 859 for test. We extract $D = 128$ wavelet features (Oliva and Torralba, 2001) from each image. In some experiments we also use the NUS-WIDE-LITE subset of this dataset, containing $N = 27\,807$ images for training and 27 807 images for test. (3) SIFT-1M (Jégou et al., 2011) contains $N = 1\,000\,000$ training high-resolution color images and 10 000 test images, each represented by $D = 128$ SIFT features.

We report precision and recall (%) in the test set using as true neighbors the K nearest images in Euclidean distance in the original space, and as retrieved neighbors in the binary space we either use the k nearest images in Hamming distance, or the images within a Hamming distance r (if no images satisfy the latter, we report zero precision). We also compare algorithms using our entropy-based measure of code utilization described in section 5.

Our experiments evaluate the effectiveness of our algorithm to minimize the BA objective and whether this translates into better hash functions (i.e., better image retrieval); its runtime and parallel speedup; and its precision and recall and code utilization compared to representative state-of-the-art algorithms.

6.1 How much does respecting the binary constraints help?

We focus purely on the BA objective function (reconstruction error) and study the gain obtained by the MAC optimization, which respects the binary constraints, over the suboptimal, “filter” approach of relaxing the constraints (i.e., PCA) and then binarizing the result by thresholding at 0 (tPCA) or by optimal rotation (ITQ). To compute the reconstruction error for tPCA and ITQ we find the optimal mapping $\mathbf{f}$ given their binary codes. We use the NUS-WIDE-LITE subset of the NUS-WIDE dataset. We initialize BA from AGH and BFA from tPCA and use alternating optimization in the $\mathbf{Z}$ steps. We search for $K = 50$ true neighbors and report results over a range of $L = 8$ to 32 bits in fig. 3. We can see that BA dominates all other methods in reconstruction error, as expected, and also in precision, as one might expect. tPCA is consistently the worst method by a significant margin, while ITQ and BFA are intermediate. Hence, the more we respect the binary constraints during the optimization, the better the hash function. Further experiments below consistently show that the BA precision significantly increases over the (AGH) initialization and is leading or competitive over other methods.

6.2 Z-step: alternating optimization and initialization

We study the MAC optimization if doing an inexact $\mathbf{Z}$ step by using alternating optimization over groups of g bits. Specifically, we study the effect on the number of iterations and runtime of the group size g and of

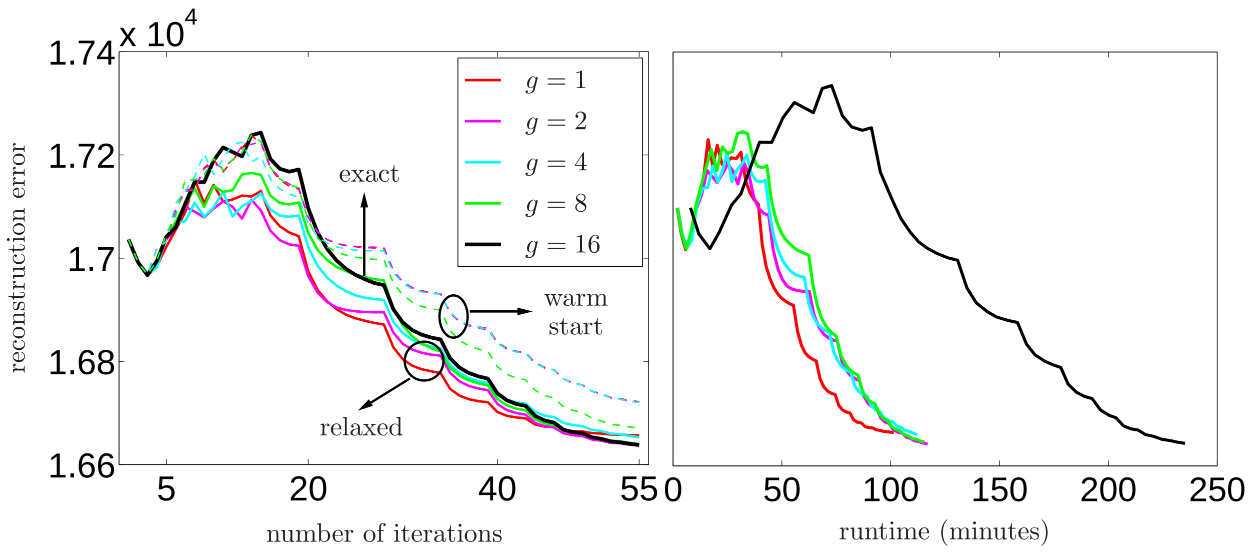


Figure 4: Iterations (*left*) and runtime (*right*) of the MAC optimization of the BA objective function (at each iteration, we run one $\mathbf{Z}$ and $(\mathbf{f}, \mathbf{h})$ step). In the $\mathbf{Z}$ step, we use alternating optimization in g-bit groups ($g = 16$ means exact optimization), and a warm-start vs relaxed initialization of $\mathbf{Z}$.

the initialization (warm-start vs relaxed QP). Fig. 4 shows the results in the CIFAR dataset using $L = 16$ bits (so using $g = 16$ gives an exact optimization), without using a validation-based stopping criterion (so we do optimize the training objective).

Surprisingly, the warm-start initialization leads to worse BA objective function values than the binarized relaxed one. Fig. 4(left) shows the dashed lines (warm-start for different g) are all above the solid lines (relaxed for different g). The reason is that, early during the optimization, the codes $\mathbf{Z}$ undergo drastic changes from one iteration to the next, so the warm-start initialization is farther from a good optimum than the relaxed one. Late in the optimization, when the codes change slowly, the warm-start does perform well. The relaxed initialization resulting optima are almost the same as using the exact binary optimization.

Also surprisingly, different group sizes g eventually converge to almost the same result as using the exact binary optimization if using the relaxed initialization. (If using warm-start, the larger g the better the result, as one would expect.) Likewise, in fig. 3, if using alternating optimization in the $\mathbf{Z}$ step rather than enumeration, the curves for BA and BFA barely vary. But, of course, the runtime per iteration grows exponentially on g (middle panel).

Hence, it appears that using faster, inexact $\mathbf{Z}$ steps does not impair the model learnt, and we settle on $g = 1$ with relaxed initialization as default for all our remaining experiments (unless we use $L < 16$ bits, in which case we simply use enumeration).

6.3 Parallel processing

Fig. 5 shows the BA training time speedup achieved with parallel processing, in CIFAR with $L = 16$ bits. We use the Matlab Parallel Processing Toolbox with up to 12 processors and simply replace “for” with “parfor” loops so each iteration (over points in the $\mathbf{Z}$ step, over bits in the $\mathbf{h}$ step) is run in a different processor. We observe a nearly perfect scaling for this particular problem. Note that, the larger the number of bits, the larger the parallelization ability in the $\mathbf{h}$ step. As a rough indication of runtimes for BA, training the 50 000 CIFAR images and 161 789 NUS-WIDE images using $L = 32$ bits with alternating optimization in the $\mathbf{Z}$ step takes 20’ and 50’, respectively (in a 4-core laptop).

6.4 Schedule of μ and initial Z

Since the objective is nonconvex, our result does depend on the initial codes $\mathbf{Z}$ (in the first iteration of the MAC algorithm), but we are guaranteed to improve or leave unchanged the precision (in the validation set)

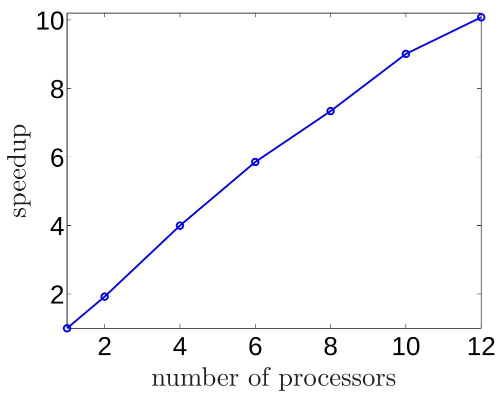


Figure 5: Parallel processing: speedup of the training algorithm as a function of the number of processors (CIFAR dataset with $L = 16$ bits).

of the codes produced by any algorithm. We have observed that initializing from AGH (Liu et al., 2011) tends to produce best results overall, so we use this in all the experiments. Using ITQ (Gong et al., 2013) produces also good results (occasionally better but generally somewhat worse than AGH), and is a simpler and faster option if so desired. We initialize BFA from tPCA, since this seems to work best.

In order to be able to use a fixed schedule, we make the data zero-mean and rescale it so the largest feature range is 1. This does not alter the Euclidean distances and normalizes the scale. We start with $\mu_1 = 0.01$ and double it after each iteration (one $\mathbf{Z}$ and $(\mathbf{f}, \mathbf{h})$ step). As noted in section 4, the algorithm will skip μ values that do not improve the precision in the validation set, and will stop at a finite μ value (past which no further changes occur).

It is of course possible to tweak all these settings (μ schedule and initial $\mathbf{Z}$) and obtain better results, but these defaults seem robust.

6.5 Comparison with other algorithms in image retrieval using precision/recall

We compare BA and BFA with the following algorithms: thresholded PCA (tPCA), Iterative Quantization (ITQ) (Gong et al., 2013), Spectral Hashing (SH) (Weiss et al., 2009), Kernelized Locality-Sensitive Hashing (KLSH) (Kulis and Grauman, 2012), AnchorGraph Hashing (AGH) (Liu et al., 2011), and Spherical Hashing (SPH) (Heo et al., 2012). Note several of these learn nonlinear hash functions and use more sophisticated error functions (that better approximate the nearest neighbor ideal), while our BA uses a linear hash function and simply minimizes the reconstruction error. All experiments use the output of AGH and tPCA to initialize BA and BFA, respectively.

It is known that the retrieval performance of a given algorithm depends strongly on the size of the neighbor set used, so we report experiments with small and large number of points in the ground truth set. For NUS-WIDE, we considered as ground truth $K = 100$ and $K = 1\,500$ neighbors of the query point, and as set of retrieved neighbors, we retrieve either k nearest neighbors ($k = 100$, 1500) or neighbors within Hamming distance r ($r = 1$, 2). Fig. 6 shows the results. For ANNSIFT-1M, we considered ground truth $K = 10\,000$ neighbors and set of retrieved neighbors for $k = 10\,000$ or $r = 1$ to 3. Fig. 7 shows the results. All curves are the average over the test set. We also show precision and recall results for the CIFAR dataset (fig. 8) as well as top retrieved images for sample query images in the test set (fig. 9), and for the NUS-WIDE-LITE dataset (fig. 10), for different ground truth sizes.

Although the relative performance of the different methods varies depending on the reported set size, some trends are clear. Generally (though not always) BA beats all other methods, sometime by a significant margin. ITQ and SPH become close (sometimes comparable) to BA in CIFAR and NUS-WIDE dataset, respectively. BFA is also quite competitive, but consistently worse than BA. The only situation when the precision of BA and BFA appears to decrease is when L is large and r is small. The reason is that many test images have no neighbors at a Hamming distance of r or less and we report zero precision for them. This suggests the hash function finds a way to avoid collisions as more bits are available. In practice, one would simply increase r to retrieve sufficient neighbors.

Fig. 7 shows the results of BA using two different initializations, from the codes of AGH and ITQ, respectively. Although the initialization does affect the local optimum that the MAC algorithm finds, this optimum is always better than the initialization and it tends to give competitive results with other algorithms.

6.6 Comparison with other algorithms in image retrieval using code utilization

Fig. 11 shows the effective number of bits L_{eff} for all methods on two datasets, NUS-WIDE and ANNSIFT-1M, and should be compared with figure 6 for NUS-WIDE and figure 7 (top row, initialized from AGH) for ANNSIFT-1M. For each method, we show L_{eff} for the training set (solid line) and test set (dashed line). We also show a diagonal-horizontal line to indicate the upper bound $L_{\text{eff}} \leq \min(L, \log_2 N)$ (again, one such line for each of the training and test sets). All methods lie below this line, and the closer they are to it, the better their code utilization.

As mentioned in section 5, tPCA consistently gives the largest L_{eff}, and L_{eff} is by itself not a consistently reliable indicator of good precision/recall. However, there is a reasonably good correlation between L_{eff} and precision (with a few exceptions, notably tPCA), in that the methods display a comparable order in both measures. This is especially true if comparing L_{eff} with the precision plots corresponding to retrieving neighbors within a Hamming distance r. This is particularly clear with the ANNSIFT-1M dataset.

Methods for which the precision drops for high L (if using a small Hamming distance r to retrieve neighbors) also show L_{eff} stagnating for those L values. This is because the small number of points (hence possibly used codes) limits the entropy achievable. These methods are making good use of the available codes, having few collisions, so they need a larger Hamming distance to find neighbors. In particular, this explains the drop in precision of BA for small r in fig. 7, which we also noted earlier. If the training or test set was larger, we would expect the precision and L_{eff} to continue to increase for those same values of L.

The correlation in performance between L_{eff} and precision is also seen if comparing methods that are using the same model and (approximately) optimizing the same objective, such as the binary autoencoder reconstruction error for ITQ and MAC. The consistent improvement of MAC over ITQ in precision is seen in L_{eff} too. This further suggests that the better optimization effected by MAC is improving the hash function (in precision, and in code utilization).

We can see that if the number of points N in the dataset (training or test) is small compared to the number of available codes 2^L, then L_{eff} stagnates as L increases (since $L_{\text{eff}} \leq \log_2 N$). This is most obvious with the test set in ANNSIFT-1M, since it is small. Together with the precision/recall curves, L_{eff} can be used to determine the number of bits L to use with a given database.

The leftmost plot of fig. 11 shows (for selected methods on NUS-WIDE) the actual code distribution as a histogram. That is, we plot the number of high-dimensional vectors c_i that map to code i, for each binary code $i \in \{0, \dots, 2^L - 1\}$ that is used (i.e., that has at least one vector mapping to it). This is the code distribution P of section 5 but unnormalized and without plotting the zero-count codes. The entropy of this distribution gives the L_{eff} value in the middle plot. High entropy corresponds to a large number of used codes, and uniformity in their counts.

7 Discussion

One contribution of this paper is to reveal a connection between ITQ (Gong et al., 2013) (a popular, effective hashing algorithm) and binary autoencoders. ITQ can be seen as a fast, approximate optimization of the BA objective function, using a "filter" approach (relax the problem to obtain continuous codes, iteratively quantize the codes, then fit the hash function). Our BA algorithm is a corrected version of ITQ.

Admittedly, there are objective functions that are more suited for information retrieval than the autoencoder, by explicitly encouraging distances in the original and Hamming space to match in order to preserve nearest neighbors (Weiss et al., 2009; Liu et al., 2011; Kulis and Grauman, 2012; Lin et al., 2013, 2014). However, autoencoders do result in good hash functions, as evidenced by the good performance of ITQ and our method (or of semantic hashing (Salakhutdinov and Hinton, 2009), using neural nets). The reason is that, with continuous codes, autoencoders can capture the data manifold in a smooth way and indirectly preserve distances, encouraging (dis)similar images to have (dis)similar codes—even if this is worsened to some extent because of the quantization introduced with discrete codes. Autoencoders are also faster and easier to optimize and scale up better to large datasets.

Note that, although similar in some respects, the binary autoencoder is not a graphical model, in particular it is not a stacked restricted Boltzmann machine (RBM). A binary autoencoder composes two *deterministic* mappings: one (the encoder) with binary outputs, and another (the decoder) with binary inputs. Hence, the objective function for binary autoencoders is discontinuous. The objective function for RBMs is differentiable, but it involves a normalization factor that is computationally intractable. This results in a very different optimization: a nonsmooth optimization for binary autoencoders (with combinatorial optimization in the step over the codes if using MAC), and a smooth optimization using sampling and approximate gradients for RBMs. See Vincent et al. (2010, p. 3372).

8 Conclusion and future work

Up to now, many hashing approaches have essentially ignored the binary nature of the problem and have approximated it through relaxation and truncation, possibly disregarding the hash function when learning the binary codes. The inspiration for this work was to capitalize on the decoupling introduced by the method of auxiliary coordinates to be able to break the combinatorial complexity of optimizing with binary constraints, and to introduce parallelism into the problem. Armed with this algorithm, we have shown that respecting the binary nature of the problem during the optimization is possible in an efficient way and that it leads to better hash functions, competitive with the state-of-the-art. This was particularly encouraging given that the autoencoder objective is not the best for retrieval, and that we focused on linear hash functions.

The algorithm has an intuitive form (alternating classification, regression and binarization steps) that can reuse existing, well-developed code. The extension to nonlinear hash and reconstruction mappings is straightforward and it will be interesting to see how much these can improve over the linear case. This paper is a step towards constructing better hash functions using the MAC framework. We believe it may apply more widely to other objective functions.

Acknowledgments

Work supported by NSF award IIS–1423515. We thank Ming-Hsuan Yang and Yi-Hsuan Tsai (UC Merced) for helpful discussions about binary hashing.

A Global optimality conditions for binary quadratic functions

Consider the following quadratic optimization with binary variables:

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{b}^T\mathbf{x} \quad \text{s.t.} \quad \mathbf{x} \in \{-1,1\}^n \tag{9}$$

where $\mathbf{Q}$ is a symmetric matrix of of $n \times n$. In general, this is an NP-hard problem (Garey and Johnson, 1979). Beck and Teboulle (2000) gave global optimality conditions for this problem that are simply expressed in terms of the problem's data ($\mathbf{Q}$, $\mathbf{b}$) involving only primal variables and no dual variables:

Sufficient If $\lambda_{\min}\mathbf{e} \geq \mathbf{XQXe} + \mathbf{Xb}$ then $\mathbf{x}$ is a global optimizer for (9).

Necessary If $\mathbf{x}$ is a global optimizer for (9) then $\mathbf{XQXe} + \mathbf{Xb} \leq \operatorname{diag}(\mathbf{Q})\,\mathbf{e}$.

Here, $\lambda_{\min}$ is the smallest eigenvalue of $\mathbf{Q}$, $\mathbf{e}$ is a vector of ones, $\mathbf{X} = \operatorname{diag}(\mathbf{x})$ is a diagonal matrix of $n \times n$ with diagonal entries as in vector $\mathbf{x}$, and $\operatorname{diag}(\mathbf{Q})$ is a diagonal matrix of $n \times n$ with diagonal entries q_{ii}. Intuitively, if $\mathbf{Q}$ is "smaller" than $\mathbf{b}$ then we can disregard the quadratic term and trivially solve the separable, linear term.

Jeyakumar et al. (2007) further gave additional global optimality conditions:

Sufficient If $\operatorname{diag}(\tilde{\mathbf{q}})\,\mathbf{e} \geq \mathbf{XQXe} + \mathbf{Xb}$ then $\mathbf{x}$ is a global optimizer for (9), where $\tilde{q}_i = q_{ii} - \sum_{j \neq i}^{n} |q_{ij}|,\ i = 1, \dots, n$.

Necessary If $\mathbf{x}$ is a global optimizer for (9) then $\mathbf{b}^T\mathbf{x} \leq \mathbf{0}$.

It is possible to give tighter conditions (Xia, 2009) but which are more complicated to compute.

Furthermore, Beck and Teboulle (2000) also gave conditions for the minimizer of the relaxed problem to be the global minimizer of the binary problem. Let the continuous relaxation be:

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{b}^T\mathbf{x} \quad \text{s.t.} \quad \mathbf{x} \in [-1,1]^n \tag{10}$$

and consider the case where $\mathbf{Q}$ is positive semidefinite, so this problem is convex. Then, a necessary and sufficient condition for $\mathbf{x}$ to be a solution of both (9) and (10) is:

$$\mathbf{x} \in \{-1,1\}^n \text{ is a global optimizer for (9) and (10)} \Longleftrightarrow \mathbf{XQXe} + \mathbf{Xb} \leq \mathbf{0}.$$

Furthermore, we have a relation when the solution of the binary problem is "close enough" to that of the relaxed one. If $\mathbf{x}$ is an optimizer of the relaxed problem (10) and $\mathbf{y} = \operatorname{sgn}(\mathbf{x})$ satisfies $\mathbf{YQ}(\mathbf{y} - \mathbf{x}) \leq \lambda_{\min}\mathbf{e}$ then $\mathbf{y}$ is a global optimizer of the binary problem (9) (where $\operatorname{sgn}(t) = 1$ if $t \geq 0$ and -1 if $t < 0$).

In our case, we are particularly interested in the sufficient conditions, which we can combine as

$$\min(\operatorname{diag}(\tilde{\mathbf{q}}), \lambda_{\min})\mathbf{e} \geq \mathbf{XQXe} + \mathbf{Xb} \Rightarrow \mathbf{x} \text{ is a global minimizer of (9).}$$

Computationally, these conditions have a comparable cost to evaluating the objective function $e(\mathbf{z}) = \|\mathbf{x} - \mathbf{f}(\mathbf{z})\|^2 + \mu\|\mathbf{z} - \mathbf{h}(\mathbf{x})\|^2$ for an L-bit vector (in the main paper). Besides, some of the arithmetic operations are common to evaluating $e(\mathbf{z})$ and the necessary and sufficient conditions for global optimality of the binary problem, so we can use the latter as a fast test to determine whether we have found the global minimizer and stop the search (when enumerating all 2^L vectors). Likewise, we can use the necessary and sufficient conditions to determine whether the solution to the relaxed problem is the global minimizer for the discrete problem, since in our $e(\mathbf{z})$ function the quadratic function is convex. Also, note that computing $\operatorname{diag}(\tilde{\mathbf{q}})$ and $\lambda_{\min}$ is done just once for all N data points in the training set, since all the problems (i.e., finding the binary codes for each data point) share the same matrix $\mathbf{A}$.

References

A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Comm. ACM*, 51(1):117–122, Jan. 2008.

A. Beck and M. Teboulle. Global optimality conditions for quadratic optimization problems with binary constraints. *SIAM Journal on Optimization*, 11(1):179–188, 2000.

M. Á. Carreira-Perpiñán. An ADMM algorithm for solving a proximal bound-constrained quadratic program. arXiv:1412.8493 [math.OC], Dec. 29 2014.

M. Á. Carreira-Perpiñán and W. Wang. Distributed optimization of deeply nested systems. arXiv:1212.5921 [cs.LG], Dec. 24 2012.

M. Á. Carreira-Perpiñán and W. Wang. Distributed optimization of deeply nested systems. In S. Kaski and J. Corander, editors, *Proc. of the 17th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2014)*, pages 10–19, Reykjavik, Iceland, Apr. 22–25 2014.

T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng. NUS-WIDE: A real-world web image database from National University of Singapore. In *Proc. ACM Conf. Image and Video Retrieval (CIVR'09)*, Santorini, Greece, July 8–10 2009.

P. L. Combettes and J.-C. Pesquet. Proximal splitting methods in signal processing. In H. H. Bauschke, R. S. Burachik, P. L. Combettes, V. Elser, D. R. Luke, and H. Wolkowicz, editors, *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, Springer Series in Optimization and Its Applications, pages 185–212. Springer-Verlag, 2011.

P. Diaconis and D. Freedman. Asymptotics of graphical projection pursuit. *Annals of Statistics*, 12(3): 793–815, Sept. 1984.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *J. Machine Learning Research*, 9:1871–1874, Aug. 2008.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

L. Getoor and T. Scheffer, editors. *Proc. of the 28th Int. Conf. Machine Learning (ICML 2011)*, Bellevue, WA, June 28 – July 2 2011.

G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, third edition, 1996.

Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A Procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, Dec. 2013.

J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon. Spherical hashing. In *Proc. of the 2012 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'12)*, pages 2957–2964, Providence, RI, June 16–21 2012.

H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 33(1):117–128, Jan. 2011.

V. Jeyakumar, A. M. Rubinov, and Z. Y. Wu. Non-convex quadratic minimization problems with quadratic constraints: Global optimality conditions. *Math. Prog.*, 110(3):521–541, Sept. 2007.

R. Kohavi and G. H. John. The wrapper approach. In H. Liu and H. Motoda, editors, *Feature Extraction, Construction and Selection. A Data Mining Perspective*. Springer-Verlag, 1998.

A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Dept. of Computer Science, University of Toronto, Apr. 8 2009.

A. Krizhevsky and G. E. Hinton. Using very deep autoencoders for content-based image retrieval. In *Proc. of the 19th European Symposium on Artificial Neural Networks (ESANN 2011)*, Bruges, Belgium, Apr. 27–29 2011.

B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 22, pages 1042–1050. MIT Press, Cambridge, MA, 2009.

B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 34(6):1092–1104, June 2012.

G. Lin, C. Shen, D. Suter, and A. van den Hengel. A general two-step approach to learning-based hashing. In *Proc. 14th Int. Conf. Computer Vision (ICCV'13)*, pages 2552–2559, Sydney, Australia, Dec. 1–8 2013.

G. Lin, C. Shen, Q. Shi, A. van den Hengel, and D. 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)*, pages 1971–1978, Columbus, OH, June 23–28 2014.

W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In Getoor and Scheffer (2011), pages 1–8.

J.-J. Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. *C. R. Acad. Sci. Paris Sér. A Math.*, 255:2897–2899, 1962.

B. Neyshabur, N. Srebro, R. Salakhutdinov, Y. Makarychev, and P. Yadollahpour. The power of asymmetry in binary hashing. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 26, pages 2823–2831. MIT Press, Cambridge, MA, 2013.

J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006.

M. Norouzi and D. Fleet. Minimal loss hashing for compact binary codes. In Getoor and Scheffer (2011).

A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *Int. J. Computer Vision*, 42(3):145–175, May 2001.

R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM J. Control and Optim.*, 14 (5):877–898, 1976.

R. Salakhutdinov and G. Hinton. Semantic hashing. *Int. J. Approximate Reasoning*, 50(7):969–978, July 2009.

A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *Proc. of the 2008 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'08)*, Anchorage, AK, June 23–28 2008.

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Machine Learning Research*, 11:3371–3408, 2010.

Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In D. Koller, Y. Bengio, D. Schuurmans, L. Bottou, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 21, pages 1753–1760. MIT Press, Cambridge, MA, 2009.

P. Whittle. On principal components and least square methods of factor analysis. *Skand. Aktur. Tidskr.*, 36:223–239, 1952.

Y. Xia. New optimality conditions for quadratic optimization problems with binary constraints. *Opt. Lett.*, 3(2):253–263, Mar. 2009.

S. X. Yu and J. Shi. Multiclass spectral clustering. In *Proc. 9th Int. Conf. Computer Vision (ICCV'03)*, pages 313–319, Nice, France, Oct. 14–17 2003.

D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. In *Proc. of the 33rd ACM Conf. Research and Development in Information Retrieval (SIGIR 2010)*, pages 18–25, Geneva, Switzerland, July 19–23 2010.

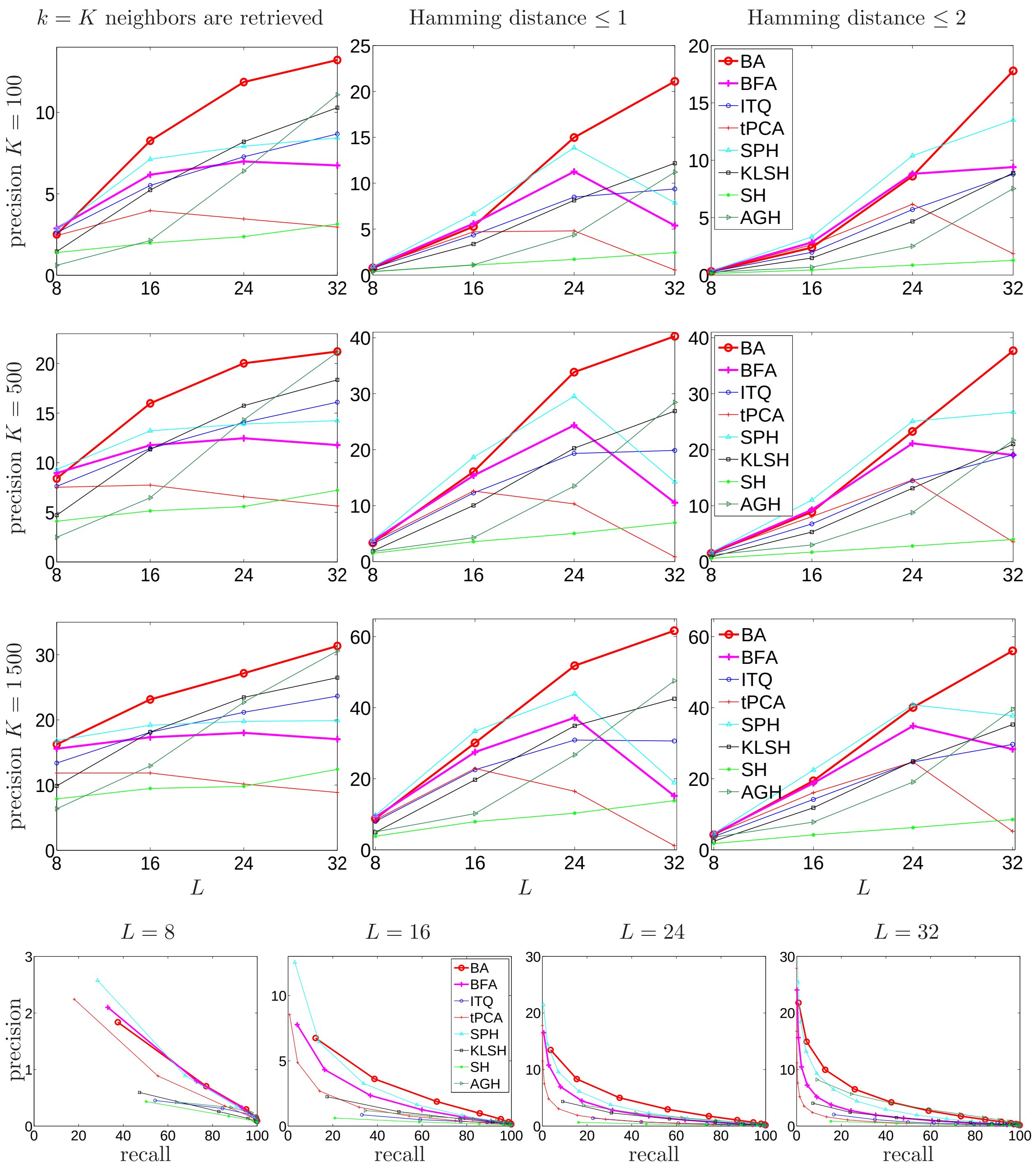


Figure 6: Precision and precision/recall in NUS-WIDE dataset. *Top block*: precision using as ground truth $K = 100$ (*top panels*), 500 (*middle panels*) and 1 500 (*bottom panels*) nearest images to the query image in the training set. Retrieved neighbors: k nearest images to the query image, where k is equal to the size of the ground-truth set (*left*), or neighbors at Hamming distance $\leq r = 1$ (*middle*) or $\leq r = 2$ (*right*), searching the training set binary codes. *Bottom block*: precision/recall using $L = 8$ to 32 bits. Ground truth: $K = 100$ nearest images to the query image in the training set. Retrieved neighbors: training images at Hamming distance $\leq r$ of the query in binary space. Within each plot, the markers left to right along each curve show $r = 0, 1, 2 \dots$. Test points not returning any neighbors are ignored in the precision/recall plots.

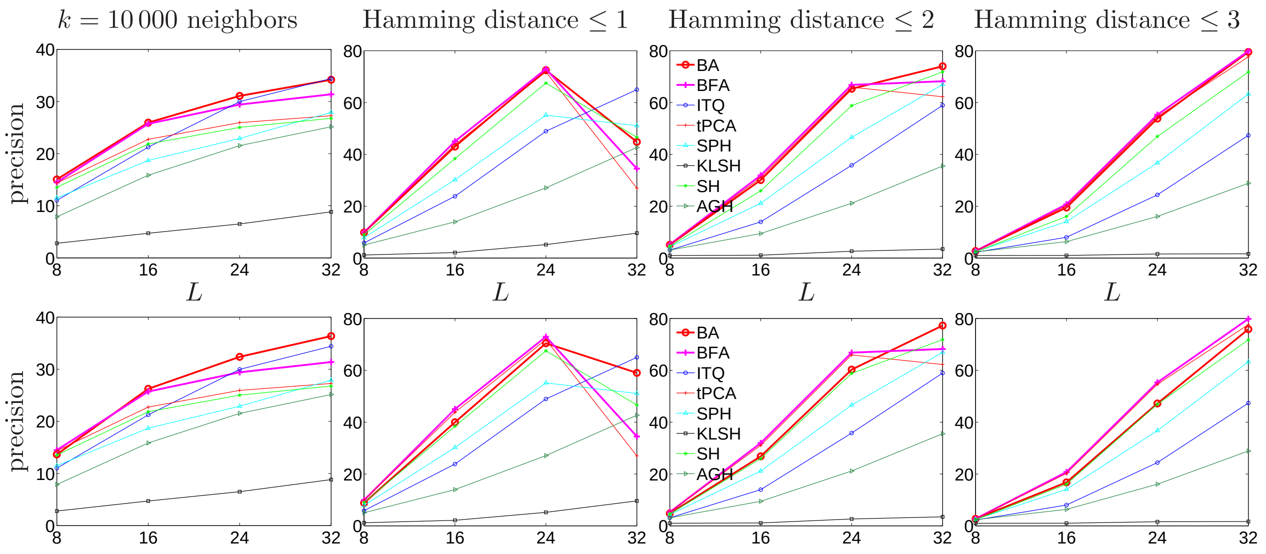
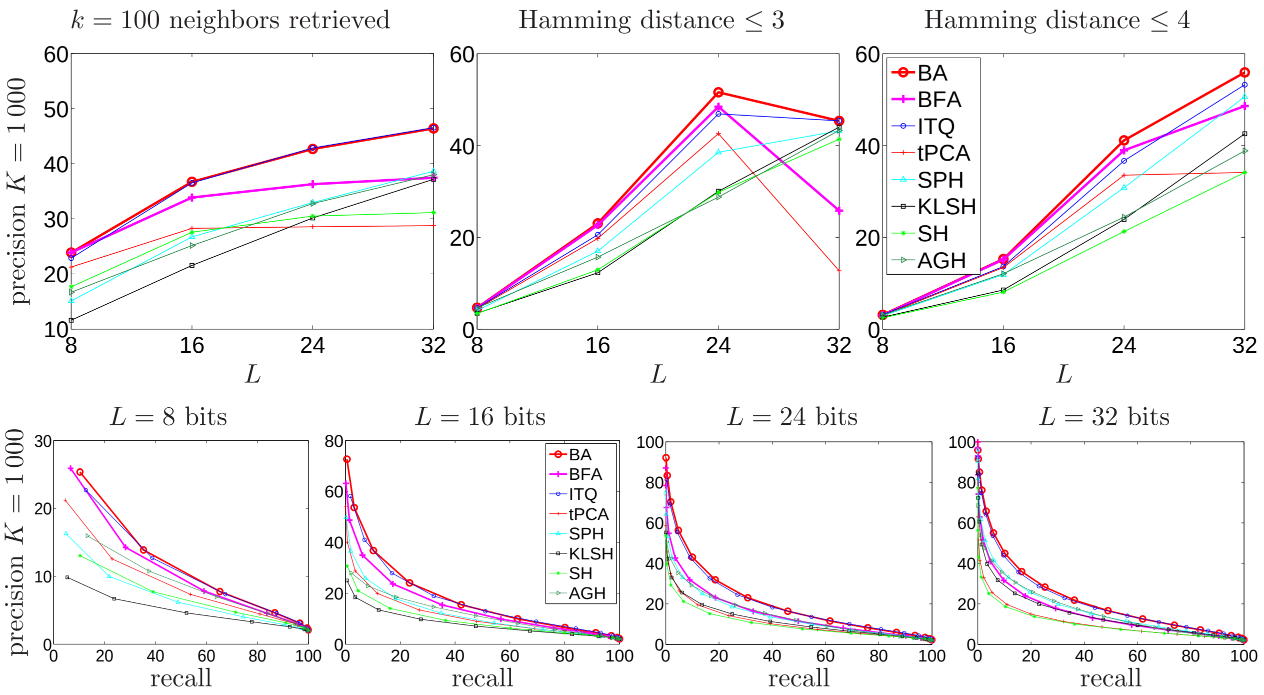


Figure 7: Precision in ANNSIFT-1M using $L = 8$ to 32 bits. Like fig. 6 (top block) but with ground truth $K = 10\,000$ and retrieved neighbors $k = 10\,000$ or Hamming distance $\leq r = 1$ to 3. BFA is initialized with tPCA. BA is initialized with AGH (*top panel*) and ITQ (*bottom panel*).



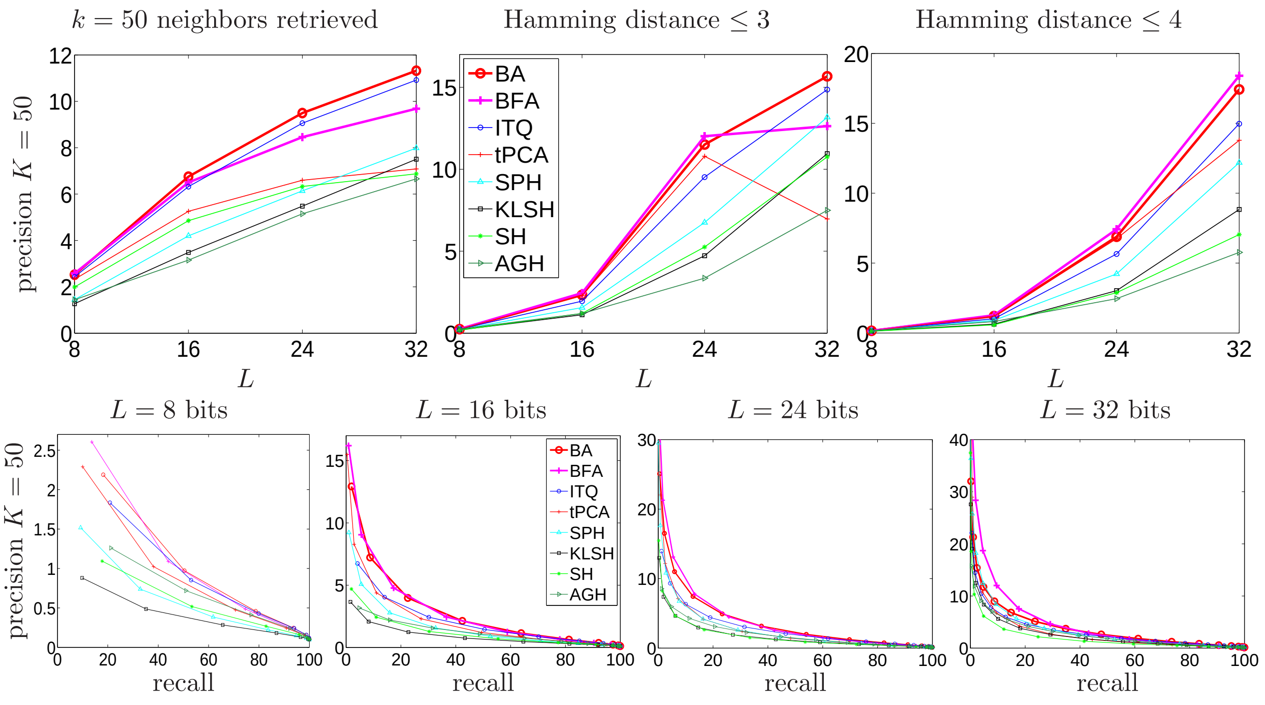


Figure 8: Precision and precision/recall in CIFAR dataset (plotted as in fig. 6). Ground truth: $K = 1\,000$ (*top block*) and $K = 50$ (*bottom block*) nearest images to the query image in the training set. *Top panels within each block*: precision depending on the retrieved set (k nearest neighbors in Hamming distance, or images at Hamming distance ≤ 3 or 4). *Bottom panels within each block*: precision/recall curves for a retrieved set of images at Hamming distance $\leq r$, using $L = 8$ to 32 bits.

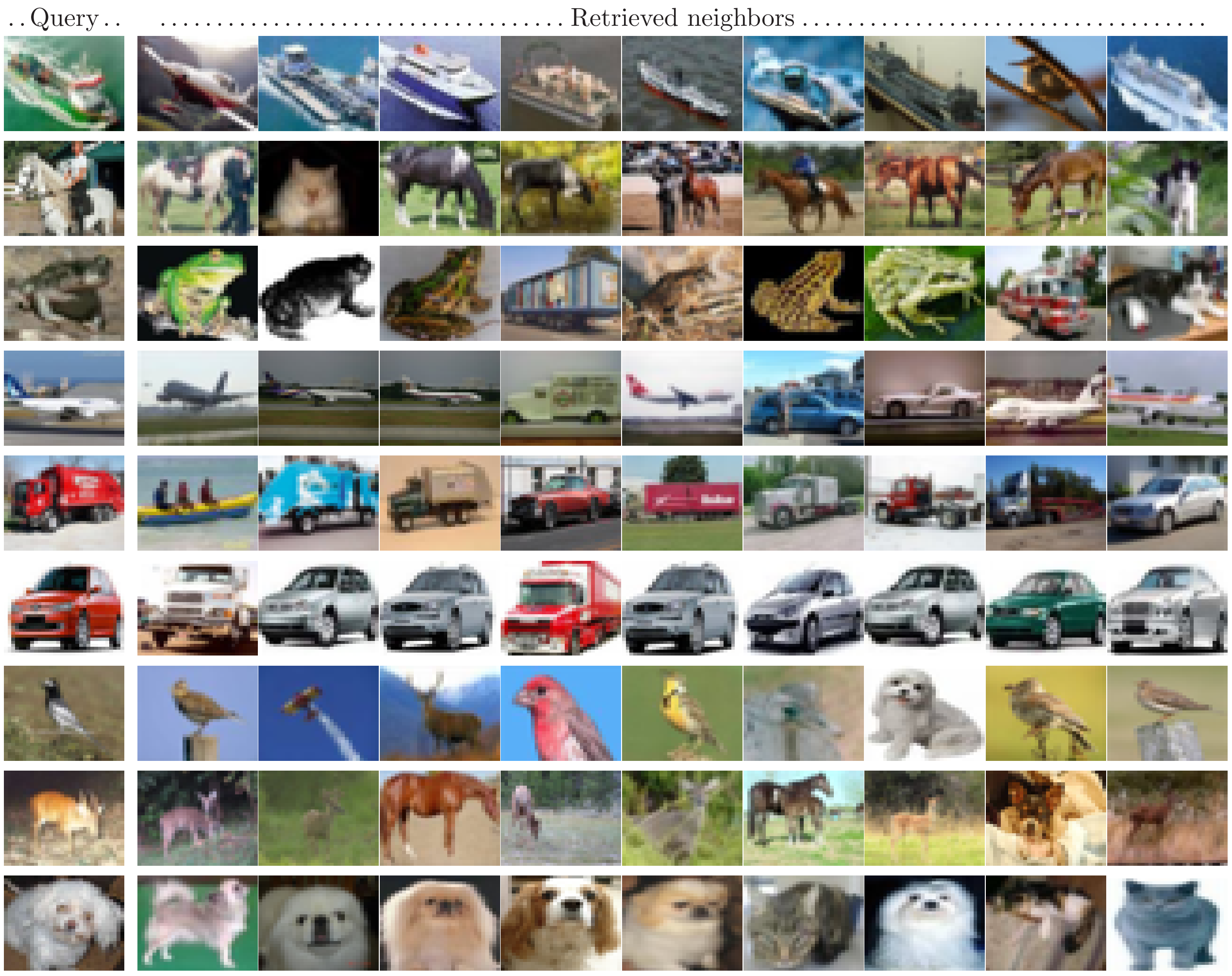

. . Query . .
. . . . Retrieved neighbors . . . .

Figure 9: Top retrieved images for sample test images from the CIFAR dataset using $L = 32$ bits.

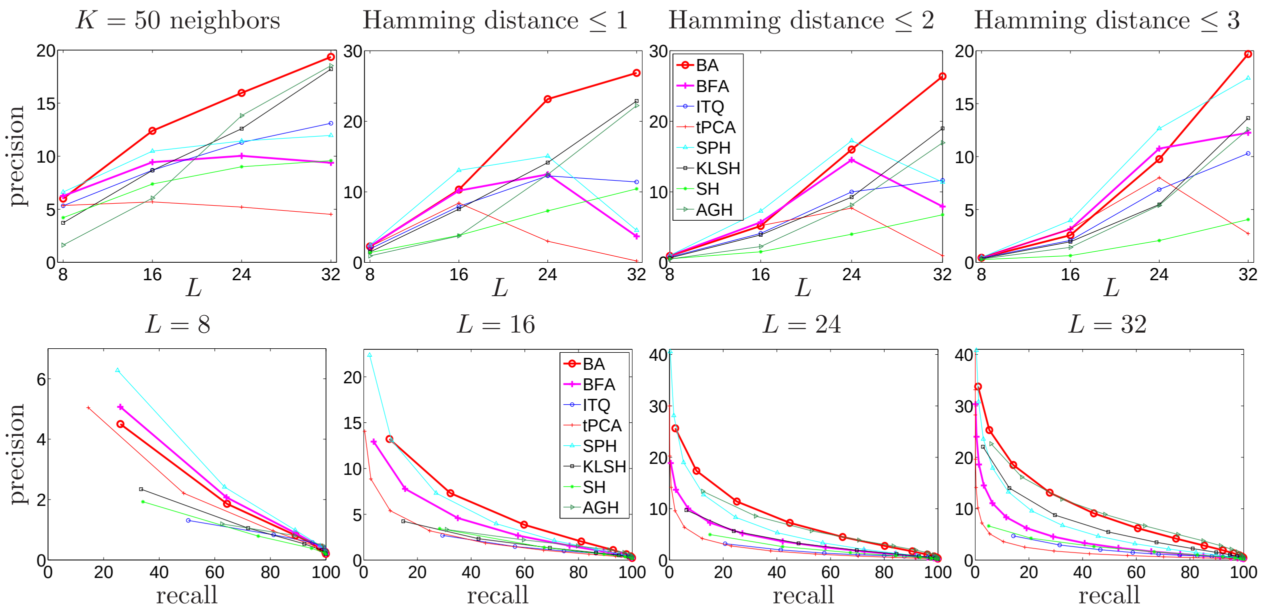


Figure 10: Precision and precision/recall in NUS-WIDE-LITE dataset (plotted as in fig. 6). Ground truth: $K = 50$ nearest images to the query image in the training set. *Top*: precision depending on the retrieved set ($k = 50$ nearest neighbors in Hamming distance, or images at Hamming distance ≤ 1 or 2 or 3). *Bottom*: precision/recall curves for a retrieved set of images at Hamming distance $\leq r$, using $L = 8$ to 32 bits.

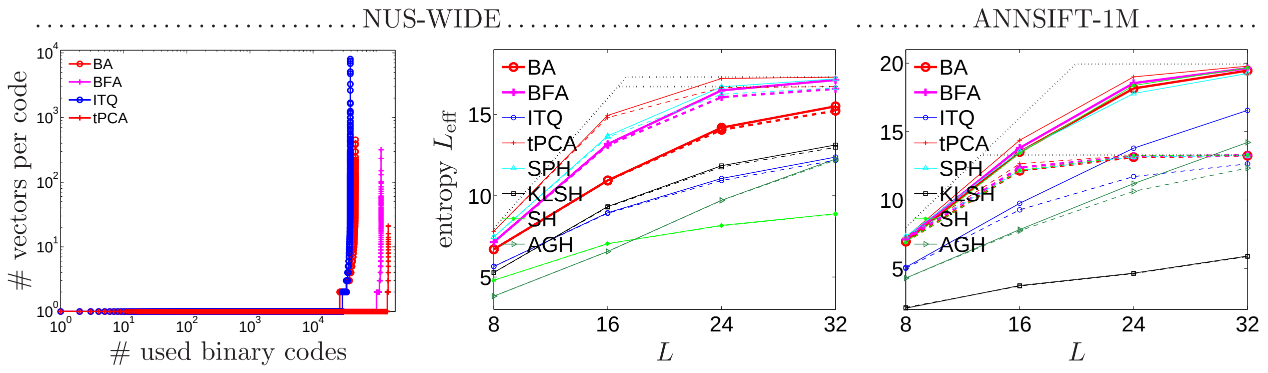


Figure 11: Code utilization in effective number of bits L_{eff} (entropy of code distribution) of different hashing algorithms, using $L = 8$ to 32 bits, for the NUS-WIDE (*left*) and ANNSIFT-1M (*right*) datasets. The plots correspond to the codes obtained by the algorithms in figures 6 and 7 (top row, initialized from AGH), respectively. The hashing algorithms are color-coded, with solid lines for the training set and dashed lines for the test set. The two diagonal-horizontal black dotted lines give the upper bound (maximal code utilization) on L_{eff} of any algorithm for the training and test sets. The leftmost plot, for NUS-WIDE, shows the histogram of high-dimensional vector counts per binary code, sorted by count, for selected methods. Only codes with at least one vector mapping to them are shown. The entropy of this distribution gives the L_{eff} value in the middle plot.