Proceedings of the 2015 INFORMS Workshop on Data Mining and Analytics
M. G. Baydogan, S. Huang, A. Oztekin, eds.

Hashing with Binary Autoencoders

Ramin Raziperchikolaei
EECS
University of California, Merced
Merced, CA
rraziperchikolaei@ucmerced.edu

Miguel Á. Carreira-Perpiñán EECS
University of California, Merced
Merced, CA
mcarreira-perpinan@ucmerced.edu

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 show the resulting hash function outperforms or is competitive with state-of-the-art methods for binary hashing.

Keywords: fast image retrieval, binary hashing, binary autoencoder, method of auxiliary coordinates

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 [9] 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 [9]. 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 this joint optimization, respecting the binary constraints during training, can actually be done reasonably efficiently. The idea is to use the recently proposed *method of auxiliary coordinates (MAC)* [2, 3]. 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. The idea of MAC has also been used in hashing to optimize the affinity based objective functions jointly over codes and hash functions [14]. In our case, MAC 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.

2 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 $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$:

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

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):

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

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., $\mathbf{Z} \in \mathbb{R}^{L \times N}$, 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)*. 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 (1) jointly over $\mathbf{f}$ and $\mathbf{h}$.

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

We use the recently proposed *method of auxiliary coordinates (MAC)* [2, 3]. 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. Recall eq. (1), 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:

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

Note the codes are binary. We now apply the quadratic-penalty method (it is also possible to apply the augmented Lagrangian method instead; [13]) and minimize the following objective function while progressively increasing μ, so the constraints are eventually satisfied:

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 $\mu = 0 < \mu_1 < \cdots < \mu_\infty$
  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**
    $\mathbf{z}_n \leftarrow \operatorname{argmin}_{\mathbf{z}_n \in \{0,1\}^L} \|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\|^2 + \mu \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2$
  if no change in $\mathbf{Z}$ and $\mathbf{Z} = \mathbf{h}(\mathbf{X})$ **then** stop
return $\mathbf{h}$, $\mathbf{Z} = \mathbf{h}(\mathbf{X})$

Figure 1: Binary autoencoder MAC algorithm.

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

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, 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. 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 for 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 in [2] because of the binary nature of the problem. Instead, it is easy to 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 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. 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 20 iterations, when no further changes to the parameters occur.

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 $(\mathbf{Z}, \mathbf{X})$, whose solution 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.

h step: This has the following form:

$$\min_{\mathbf{h}} \sum_{n=1}^{N} \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2 = \min_{\mathbf{W}} \sum_{n=1}^{N} \|\mathbf{z}_n - \sigma(\mathbf{W}\mathbf{x}_n)\|^2 = \sum_{l=1}^{L} \min_{\mathbf{w}_l} \sum_{n=1}^{N} (z_{nl} - \sigma(\mathbf{w}_l^T \mathbf{x}_n))^2. \tag{5}$$

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. This surrogate loss has the advantage of making the solution unique and generalizing better to test data. 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 [5] 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.

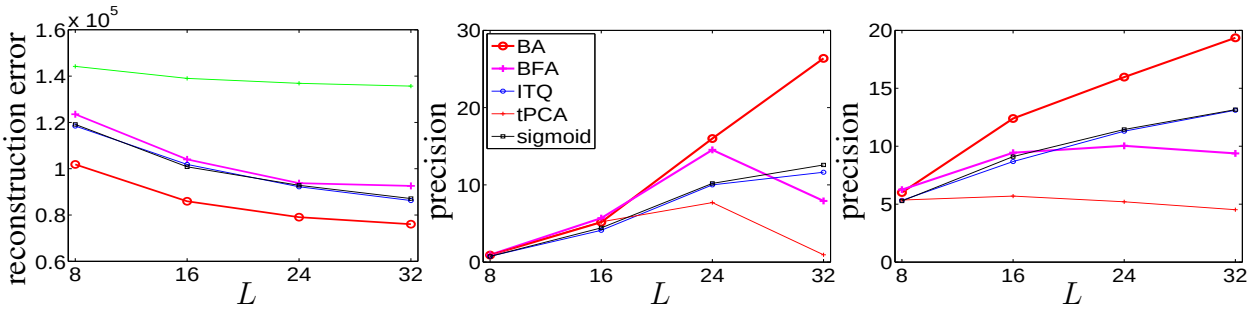

Figure 2: Wrapper vs filter optimization: BA objective function (*left*) and precision using neighbors within Hamming distance $r = 2$ (*middle*) and using $k = 50$ nearest neighbors (*right*).

Z step: From eq. (4), this is a binary optimization on NL variables, but it separates into N independent optimizations each on only L variables (where we omit the index n):

$$\min_{\mathbf{z}} e(\mathbf{z}) = \|\mathbf{x} - \mathbf{f}(\mathbf{z})\|^2 + \mu\|\mathbf{z} - \mathbf{h}(\mathbf{x})\|^2 \text{ s.t. } \mathbf{z} \in \{0,1\}^L. \tag{6}$$

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.

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. We have developed an ADMM algorithm [1] that is very simple, vectorizes very well, and reuses matrix factorizations over all N QPs.

Schedule for the penalty parameter 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 $0 < \mu_1 < \cdots < \infty$). Fortunately, setting the schedule 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. 2) In order to generalize well to unseen data, we stop iterating when the precision in a validation set decreases. This is a form of early stopping that guarantees that we improve the initial $\mathbf{Z}$.

4 Experiments

We used three datasets in our experiments, commonly used as benchmarks for image retrieval. (1) CIFAR [10] contains 50 000 training and 10 000 test 32×32 color images, each represented by $D = 320$ GIST features. (2) NUS-WIDE [4] contains $N = 161\,789$ high-resolution images for training and 107 859 for test, each image represented by $D = 128$ wavelet features. (3) SIFT-1M [8] 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.

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). We also try relaxing the step function to a sigmoid during training (with back-propagation using minibatches of 500 points) as in [15]. To compute the reconstruction error for tPCA and ITQ we

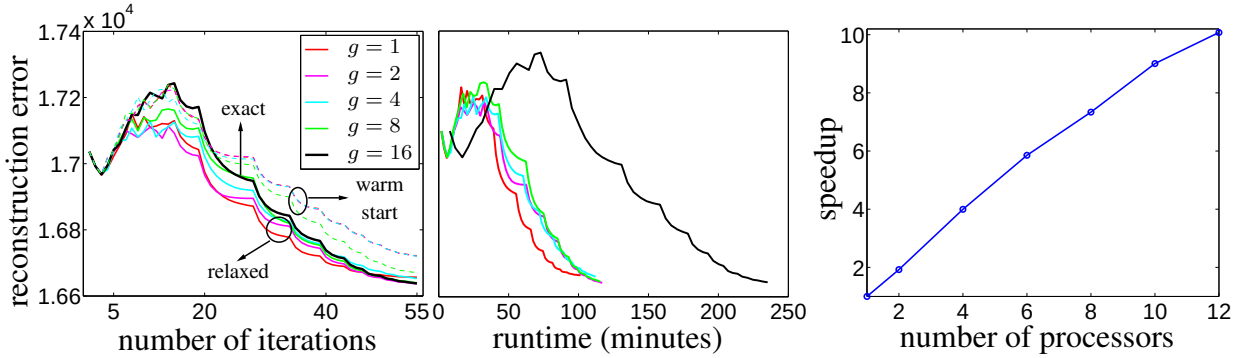

Figure 3: Iterations (*left*) and runtime (*middle*) of the MAC optimization of the BA objective function (at each iteration, we run one $\mathbf{Z}$ and $(\mathbf{f}, \mathbf{h})$ step). In the $\mathbf{Z}$ step, we use alternating optimization in g-bit groups, and a warm-start vs relaxed initialization of $\mathbf{Z}$. *Right*: Parallel processing

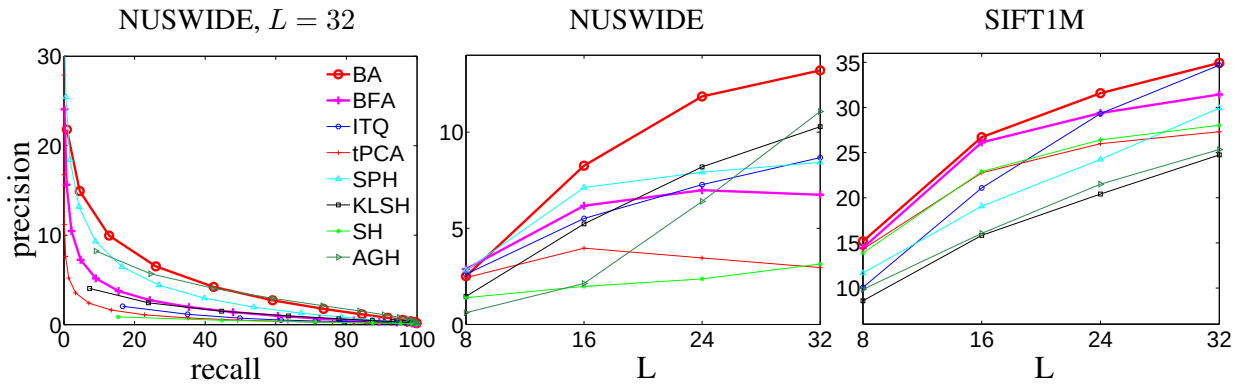

Figure 4: Precision and Precision/recall in NUS-WIDE and SIFT1M datsets. Ground truth: $K = 100$ and $K = 10\,000$ nearest images to the query image in the training set of NUS-WIDE and SIFT1M, respectively. Retrieved neighbors: k nearest images to, or images at Hamming distance r of the query, searching the training set binary codes.

find the optimal mapping $\mathbf{f}$ given their binary codes. We use the NUS-WIDE-LITE subset of the NUS-WIDE dataset, containing $N = 27\,807$ NUS-WIDE images for training and $27\,807$ images for test. We initialize BA from AGH [12] 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. 2. We can see that BA dominates all other methods in reconstruction error, as expected, and also in precision, as one might expect. Hence, the more we respect the binary constraints during the optimization, the better the hash function.

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 the initialization (warm-start vs relaxed QP). Fig. 3 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. 3 (left) shows the dashed lines (warm-start for different g) are all above the solid lines (relaxed for different g). Also surprisingly, different group sizes g eventually converge to almost the same result as using the exact binary optimization if using the relaxed initialization.

Parallel processing Fig. 3, right column, 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. 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).

Comparison with other algorithms in image retrieval We compare BA and BFA with the following algorithms: thresholded PCA (tPCA), Iterative Quantization (ITQ) [6], Spectral Hashing (SH) [16], Kernelized Locality-Sensitive Hashing (KLSH) [11], AnchorGraph Hashing (AGH) [12], and Spherical Hashing (SPH) [7]. 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.

For NUS-WIDE, we considered as ground truth $K = 100$ neighbors of the query point, and as set of retrieved neighbors, we retrieve either $k = 100$ nearest neighbors or neighbors within Hamming distance r. For ANNSIFT-1M, we considered ground truth $K = 10\,000$ neighbors and set of retrieved neighbors for $k = 10\,000$. Fig. 4 shows the results. 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 (not shown) and NUS-WIDE dataset, respectively.

5 Conclusion

Up to now, many hashing approaches have essentially ignored the binary nature of the problem and have approximated it through relaxation and truncation. 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.

References

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

[2] Miguel Á. Carreira-Perpiñán and Weiran Wang. Distributed optimization of deeply nested systems. arXiv:1212.5921 [cs.LG], December 24 2012.

[3] Miguel Á. Carreira-Perpiñán and Weiran Wang. Distributed optimization of deeply nested systems. In Samuel Kaski and Jukka Corander, editors, *Proc. of the 17th Int. Conf. AISTATS 2014)*, pages 10–19, Iceland, 2014.

[4] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yan-Tao Zheng. NUS-WIDE: A real-world web image database from National University of Singapore. In *Proc. ACM Conf. CIVR'09*, Greece, 2009.

[5] 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*, 9:1871–1874, August 2008.

[6] 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. PAMI*, 35(12):2916–2929, 2013.

[7] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. Spherical hashing. In *Proc. of the 2012 IEEE Computer Society Conf. CVPR'12*, pages 2957–2964, Providence, RI, June 16–21 2012.

[8] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 33(1):117–128, January 2011.

[9] R. Kohavi and G. H. John. *Feature Extraction, Construction and Selection. A Data Mining Perspective*. Springer-Verlag, 1998.

[10] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, 2009.

[11] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing. *IEEE Trans. PAMI 2012*.

[12] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. *Proc. ICML 2011*, 2011.

[13] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series, Springer-Verlag, 2006.

[14] Ramin Raziperchikolaei and Miguel Á. Carreira-Perpiñán. Learning hashing with affinity-based loss functions using auxiliary coordinates. arXiv:1501.05352 [cs.LG], January 21 2015.

[15] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *Int. J. Approximate Reasoning*, July 2009.

[16] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. *NIPS*, MIT Press, Cambridge, MA, 2009.