Optimizing binary autoencoders using auxiliary coordinates, with application to learning binary hashing

Ramin Raziperchikolaei **Miguel Á. Carreira-Perpiñán**
EECS, University of California, Merced
http://eecs.ucmerced.edu

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 [6]. While the search in the original space would cost $\mathcal{O}(ND)$ in both time and space, 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 dataset (NL bits) can fit in the main memory of a workstation.

Many hashing papers formulate an objective function of the hash function $\mathbf{h}$ or of the binary codes that tries to capture some notion of neighborhood preservation. In their formulation $\mathbf{h}$ typically performs dimensionality reduction ($L < D$) and, as noted, it outputs binary codes ($\mathbf{h}: \mathbb{R}^D \rightarrow \{0,1\}^L$). Optimizing this is difficult, because the hash function must output binary values, hence the problem is not just generally nonconvex, but also nonsmooth. This is an NP-complete problem. While the gradients of the objective function do exist wrt $\mathbf{W}$, they are zero nearly everywhere.

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). This procedure can be seen as a "filter" approach and is suboptimal. We obtain a better optimization by optimizing the objective jointly over linear mappings and thresholds, respecting the binary constraints while learning $\mathbf{h}$; this is a "wrapper" approach [7].

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. 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.

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 $\mathbf{x}$. 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. Our *desired hash function* will be $\mathbf{h}$, and it should minimize the following problem, given a dataset of high-dimensional patterns $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$:

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

which is the usual least-squares error but where the code layer is binary. 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. 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

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 [9]) and minimize the following objective function while progressively increasing μ, so the constraints are eventually satisfied:

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

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 (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. During the iterations of the algorithm, 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.

Although a MAC algorithm can be shown to produce convergent algorithms as $\mu \to \infty$ with a differentiable objective function, we cannot use this fact here 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 3.1. *Assume the steps over* $\mathbf{h}$, $\mathbf{f}$ *are solved exactly by finding their unique global minimum. Then the MAC algorithm for the binary autoencoder stops at a finite* μ.

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, $\mathbf{f}$ and $\mathbf{Z}$ can be optimized independently from $\mathbf{h}$, but $\mathbf{h}$ must optimally fit the resulting $\mathbf{Z}$). 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.

3.1 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 $(\mathbf{Z}, \mathbf{X})$, 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.

3.2 h step

This has the following form:

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

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. Besides being easier (by reusing well-developed codes for SVMs), this surrogate loss has the advantage of making the solution unique and generalizing better to test data.

3.3 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):

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

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 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, the following proof shows that the problem, which as stated uses a matrix of $D \times L$, can be reduced to an equivalent problem using a matrix of $L \times L$.

Theorem 3.2. *Let $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{A} \in \mathbb{R}^{D\times L}$, with QR factorisation $\mathbf{A} = \mathbf{Q}\mathbf{R}$, 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 $\mathbf{z}$:*

$$\min_{\mathbf{z}\in\{0,1\}^L} \|\mathbf{x} - \mathbf{A}\mathbf{z}\|^2 \quad \min_{\mathbf{z}\in\{0,1\}^L} \|\mathbf{y} - \mathbf{R}\mathbf{z}\|^2. \quad (6)$$

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 $40\times$ if using 16 bits with $D = 320$ GIST features in our experiments.

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. $L = 16$ 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 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 that is very simple, parallelizes or vectorizes very well, and reuses matrix factorizations over all N QPs.

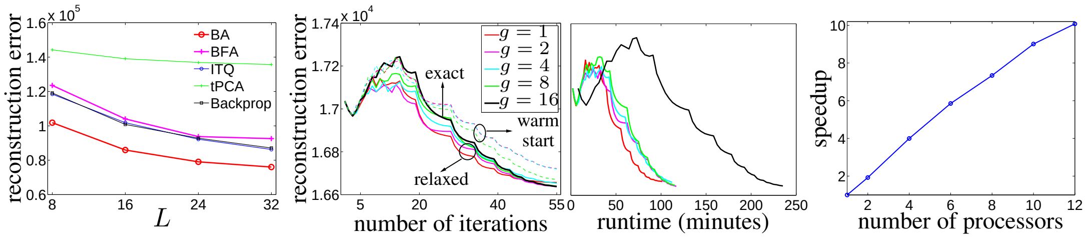

Figure 1: *First panel*: our wrapper approach (BA) vs other filter approaches. *Second and third panels*: effect of initialization and group sizes g of alternating optimization in objective function value. *Fourth panel*: increasing the number of processors leads to an almost linear speedup.

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 $\mathbf{Z}$ does not change compared to the previous $\mathbf{Z}$ step. 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.

4 Experiments

In this section we show a subset of our experiments. More results can be found in the main paper [1]. In our first experiment, 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 [5]). We also try relaxing the step function to a sigmoid during training (with backpropagation using minibatches of 500 points) as in [11]. All the methods are trained using a subset of 27 807 images from NUS-WIDE dataset [4]. Fig. 1 shows the reconstruction error as we increase the number of bits. We can see that BA dominates all other methods in reconstruction error as expected. The better optimization of BA will also lead to better image retrieval results (the retrieval results can be found in the main paper [1]).

In the following three experiments we use the CIFAR dataset [8], containing 58 000 points for training and 2 000 points for test, to train the binary autoencoder. Each image is represented by $D = 320$ GIST features [10]. In the second panel of the figure, we investigate the effect of initialization on the value of the objective function. The figure shows that the warm-start initialization leads to worse BA objective function values than the binarized relaxed one. We can see that 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.

The third panel of fig. 1 shows the effect of group sizes g of alternating optimization in the value of the objective function. As we can see, 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.)

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).

The last panel of fig. 1 shows the BA training time speedup achieved with parallel processing, 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.

References

[1] M. Á. Carreira-Perpiñán and R. Raziperchikolaei. Hashing with binary autoencoders. In *Proc. of the 2015 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'15)*, pages 557–566, Boston, MA, June 7–12 2015.

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

[3] 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.

[4] 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.

[5] 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.

[6] K. Grauman and R. Fergus. Learning binary hash codes for large-scale image search. In R. Cipolla, S. Battiato, and G. Farinella, editors, *Machine Learning for Computer Vision*, pages 49–87. Springer-Verlag, 2013.

[7] 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.

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

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

[10] 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.

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