An NMF perspective on Binary Hashing

Lopamudra Mukherjee[§], Sathya N. Ravi[†], Vamsi K. Ithapu[†], Tyler Holmes[§], Vikas Singh[†]
[†]University of Wisconsin-Madison, [§]University of Wisconsin-Whitewater

Abstract

The pervasiveness of massive data repositories has led to much interest in efficient methods for indexing, search, and retrieval. For image data, a rapidly developing body of work for these applications shows impressive performance with methods that broadly fall under the umbrella term of Binary Hashing. Given a distance matrix, a binary hashing algorithm solves for a binary code for the given set of examples, whose Hamming distance nicely approximates the original distances. The formulation is non-convex — so existing solutions adopt spectral relaxations or perform coordinate descent (or quantization) on a surrogate objective that is numerically more tractable. In this paper, we first derive an Augmented Lagrangian approach to optimize the standard binary Hashing objective (i.e., maintain fidelity with a given distance matrix). With appropriate step sizes, we find that this scheme already yields results that match or substantially outperform state of the art methods on most benchmarks used in the literature. Then, to allow the model to scale to large datasets, we obtain an interesting reformulation of the binary hashing objective as a non-negative matrix factorization. Later, this leads to a simple multiplicative updates algorithm — whose parallelization properties are exploited to obtain a fast GPU based implementation. We give a probabilistic analysis of our initialization scheme and present a range of experiments to show that the method is simple to implement and competes favorably with available methods (both for optimization and generalization).

1. Introduction

The pervasiveness of massive image data repositories with billions (and even trillions) of examples has led to a renewed interest in efficient algorithms for search and indexing. These datasets correspond to surveillance data, text, video sequences, measurements from scientific experiments, gene expression profiles and photo collections — in each case, the data is very high dimensional and efficient storage and retrieval allows important downstream applications to operate seamlessly. Driven by these requirements, research in the last decade has focused on both compact representations of the data to mitigate dependence on its native dimensionality as well as *approximate* forms of nearest neighbor schemes since a linear scan is impractical in most, if not all, practical deployments.

A simple solution which is (partly) cognizant of both the space and time considerations in the above setting is dimensionality reduction. Clearly, by mapping (or embedding) the distribution from the native space D to lower dimensions, $d \ll D$, significantly reduces the storage requirements. Further, even if a linear (or nearest neighbor) search were employed, its running time may only be weakly dependent on D. While this strategy is sometimes sufficient, the literature suggests that retrieval times can be substantially improved if the embedding were *binary*. The Hamming distance between a query point and an item in the database can now be computed via logical operations (such as XOR). This allows a naïve search, even on a massive dataset, to run within a few seconds. Also, allocating 32 or 64 bits per item (e.g., an image), one can fit a billion-sized dataset in the main memory of a moderately priced workstation. These observations have led to a nice set of results which adapt the rich theory of Locality Sensitive Hashing (LSH) [13] to this application. LSH obtains dimensionality reduction using Random projections [5]. By the Johnson-Lindenstrauss lemma, original distances between the examples are preserved (up to some distortion) with high probability. Now, as the number of bits grows, LSH offers increasingly better fidelity with the original raw distances. Recent work has also derived kernelized versions of LSH [17], which makes the ideas applicable in an even broader setting.

Despite the various advantages of LSH, empirical results suggest that with a restricted number of bits (usually less than a hundred), LSH does not perform very well. In other words, to be practically useful, the number of bits may have to be large which seriously limits its applicability in problems in computer vision [12]. An elegant solution to this problem is to use learning within hashing, that is, instead of treating the hashing function as a black-box, several approaches *learn* the item-wise code from the data either in a supervised or unsupervised setting [12]. The core goal in most of these proposals is to derive a specific hashing code whose Hamming distances reflect the relationships given by

a semantic distance matrix or class labels obtained using domain knowledge. Such strategies significantly outperforms a vanilla LSH scheme and yield state of the art results for computer vision and NLP applications [19]. From a practical point of view, learning binary embeddings offers yet another key advantage. Observe that calculating distances between image or text data directly, as if their native representations were points in a vector space is often meaningless. One often performs extensive pre-processing to express images as feature descriptors or graphs and text may be encoded as a distribution over topics. Such objects may not support standard arithmetic operations such as addition and multiplication. So, calculation of 'distances' between them typically involve an oracle which computes graph correlation and/or geodesics or divergence measures between probability distributions over topics [3]. Conceptually, it seems logical to "learn" a binary embedding that best preserves *these domain-specific affinities*, instead of finding a lower dimensional embedding that is low-distortion merely in terms of Euclidean distance, a quantity which is inconsistent with the geometry of the space of these objects.

Related Work. Distance preserving hashing (and binary embedding) is a very well studied problem in theoretical computer science, coding theory and machine learning. To keep our review concise, we will restrict our discussion to papers that deal with *learning* binary hashing functions.
Semantic hashing [30] uses a deep generative model (Restricted Boltzmann machine) which is learnt via pretraining followed by a fine-tuning step to obtain the hashing function, which preserves semantic distances in Hamming space. Matches to a query vector are found by exploring a Hamming ball around the original vector. The method works well unless the number of bits d, is large, when perturbations for a given radius become expensive. Spectral hashing [35] uses the eigenfunctions of the Laplacian to capture the variance of the data, which is then assigned additional bits in the encoding. The semi-supervised hashing algorithm minimizes a loss function defined on the labeled data, but similar to spectral hashing optimizes a variance based measure for the labeled (plus the unlabeled) examples. Kulis' BRE algorithm [16] performs coordinate descent to minimize the difference between the Hamming distances and the original Euclidean distances whereas [27] uses a formulation similar to [16] but with a different loss function. Using PCA as a starting point, [11, 10] minimize the quantization error of mapping the (reduced dimensionality) real-valued data to vertices of the binary hypercube using a Procrustes type algorithm. Lin et al. [21] argue for training the hashing function one bit at a time, to allow dissimilar objects to move apart. Ke et. al. [14] formulated the task of learning the hashing function within the boosting setting. But this algorithm requires specification of example pair relationships, which may become prohibitive. Other related works include parameter sensitive hashing [31] which is an interesting variation of LSH, and asymmetric hash functions [25]. A recent work on multidimensional spectral hashing (MDSH) [34], inspired by spectral hashing, is in some sense the closest to our work. It seeks to optimize a similar cost function as us — the loss between the distance in the Hamming space and the given distance function. The bits in the final solution correspond to thresholded eigenvectors of the affinity matrix.

Maximize fidelity with *all* entries of the given distance matrix? The underlying goal in many binary hashing schemes is to make the Hamming distance of the obtained binary codes resemble the original distance matrix, in its entirety, as well as possible. On the other hand, hashing formulations studied in theoretical computer science primarily look at a *retrieval* task where the focus is slightly different. In this case, various authors [8, 34] have suggested that *only* faithfully reconstructing the smaller distances between example pairs is sufficient. Notice that this makes intuitive sense if the follow-up procedure is concerned only with (unsupervised) nearest neighbor queries, and the goal is to optimize querying time. However, in recent work, these ideas have also been used, with minor adaptation, within machine learning and computer vision where the eventual goal is to instead retrieve examples which are semantically similar to the query (e.g., objects from the same class) [15]. This change in the eventual application needs careful handling. It is easy to see that preserving *only the small distances* in an embedding is problematic here — such an approach will not bother maximizing inter-class distances of the codes, leading to poor generalization on out-of-sample examples. Essentially, the learned hashing function is biased (only by positive examples) and is not discriminative. Algorithms such as BRE, MDSH (with a large number of bits) and ITQ [16, 34, 10] indeed seek to mirror the distance matrix in its entirety, which is the approach we adopt here. This is done by solving for a matrix A_h (the hamming distance matrices obtained from the binary code), such that $|A - A_h|^2$ is minimized given a 'target' distance function A. While [16] stays close to the original idea of locality sensitive hashing and requires the hashing function to be of a certain functional form, [34] has no such restriction. In fact, this approach most closely resembles our proposal since it expresses A_{aff} (the hamming affinity matrix) as a factorization of the form, $A_{\text{aff}} = Y \Lambda Y^T$, where Y is later used to derive the binary code and Λ is a weight matrix. The solution is obtained by thresholding the eigen vectors of A_{aff}.

Our Contributions. Many results on the problem so far use eigen-vector decomposition to provide a warm start to the subsequent iterative schemes. One example of this strategy is the ITQ algorithm [10]. But very recent works [34] argue for thresholding the eigen-vectors directly. We show

with a simple toy example how this may be unsatisfactory, even in a noise-less setting. An example in the Supplement (see Figure 1) shows a target distance matrix A which is exactly equal to the Hamming distance matrix of a given code, whereas the generated code is not optimal. Of course, multiple binary codes can yield the same distance matrix; nonetheless, in some sense, this represents the *ideal* input to the problem. We construct the corresponding affinity matrix as $A_{\text{aff}} = \frac{1}{2}(2d\mathbf{1}_{n\times n} - A_h)$ as in [34] and then threshold its eigen vectors based on their sign. Even in this situation, the matrix is far from the true code. This is not a hand crafted example to show a rare occurrence. We can verify that by sampling affinity matrices which are very close estimates (modulo a small error) of affinities obtained from randomly generated codes (X), across a number of different settings, we seldom obtain the exact code. One reason for this behavior is that in computing Hamming distance (or its affinity), all bits are equally important. This is not true for eigen vectors whose importance is strictly weighted by the magnitude of eigen values. We find that such a decomposition is still useful — not for direct thresholding, but to find a basis where we obtain an initial real-valued embedding. Further, this embedding can be significantly improved in practice by a numerically stable optimization scheme, yielding state of the art results on a variety of benchmarks.

The main **contributions** of this paper are: **(i)** We present two related optimization models for binary hashing. The first is an Augmented Lagrangian approach that solves a relaxation of the original formulation. The second recasts the problem in a form very similar to the well-known Non-negative Matrix Factorization. In the latter case, the update steps are very similar to those used in widely used NMF implementations for large scale datasets; **(ii)** Our initialization (used for both approaches) is generated using a very simple rule. We show that if the data follows certain (mild) distributional assumptions, we can bound the obtained Hamming distance as a function of the original distances. Note that such results are the de-facto desiderata of any hashing framework; **(iii)** For experiments, we show substantially improved performance (for optimization and generalization performance) on benchmarks used in many state of the art algorithms for this problem. We also discuss a CUDA based implementation which is significantly faster than an non-GPU implementation, leading to significant runtime gains.

2. The Proposed Algorithm

Let $\hat{X} \in \mathbb{R}^{n\times D}$ be a high dimensional matrix representing n examples in D dimensions. Let A be the corresponding distance matrix where $A \in n \times n$. [1] Our goal is to embed $\hat{X}$ into a lower dimensional binary space $X \in n \times d$, where $D \gg d$, while preserving their relative distances to the extent possible. Since X is binary, we compute their relationship using the Hamming distance. Let A_h be the corresponding Hamming distance matrix in $\mathbb{R}^{n\times n}$, where each entry $A_h(i,j)$ gives the Hamming distance between points i and j in X. Then, our objective can be written as

$$\min \|A - A_h\|_F^2 \quad \text{s.t.} \quad A_h = \Phi(X). \tag{1}$$

Here, $\Phi(\cdot)$ is a function that provides the Hamming distance matrix of X. To represent Φ, we can write $A_h(i,j) = X_i \oplus X_j$ where X_i and X_j are row vectors of X. Note that these rows correspond to the binary embedding of points i and j and $\oplus$ denotes the XOR operation of two binary vectors. However, substituting this linear complementarity form directly into (1) yields a difficult optimization problem. Instead, we can observe that the corresponding XOR operation may be written in algebraic form as $A_h = X(1-X)^T + (1-X)X^T$. By inspection, the identity holds because an XOR operation between two binary vectors counts the number of locations where the bits are different. Therefore, the (i,j) entry of the matrix $X(1-X)^T$ simply counts the locations where X_i is 1 and X_j is 0. On the other hand, corresponding entry in $(1-X)X^T$ counts the locations where X_i is 0 and X_j is 1. Substituting the identity for A_h into (1) directly is not particularly attractive because it gives a quatric polynomial.

2.1. Augmented Lagrangian Formulation

We first present an Augmented Lagrangian based approach [7]. To do so, we first write the above conditions as a constrained optimization problem, with a slight reformulation.

Let $A_h = XE^T + EX^T - 2XX^T$, where $E \in \mathbb{R}^{n\times d}$ is a matrix of all $1s$. Essentially $(XE^T)_{ij}$ counts the number of $1s$ in X_i whereas $(EX^T)_{ij}$ does the same for X_j. Also, $(XX^T)_{ij}$ counts the number of bits where both are 1. Therefore, computing the sum of the first two terms and subtracting the intersection gives us the precise form for Hamming distance. This yields the following optimization problem,

$$\begin{aligned} \min_{X,Y} \quad & ||A - Y||_{\text{F}}^2 \\ \text{s.t.} \quad & Y = XE^T + EX^T - 2XX^T, X \in \{0,1\}^{n\times d} \end{aligned} \tag{2}$$

Note that A_h is replaced by Y for notational simplicity. We relax the binary restrictions to let $0 \le X_{ij} \le 1$ instead. Hence, we are interested in solving for a stationary point of the following optimization problem,

$$\begin{aligned} \min_{X,Y} \quad & ||A - Y||_{\text{F}}^2 \\ \text{s.t.} \quad & Y = XE^T + EX^T - 2XX^T, X \in [0,1]^{n\times d} \end{aligned} \tag{3}$$

Using the standard Augmented Lagrangian Method (ALM)

[1] "Distance" refers to any function that returns low values for near items whereas "affinity" refers to functions that return high values for near items.

[26] and denoting $\Delta = \left(XE^T + EX^T - 2XX^T\right)$,

$$\begin{aligned} \min_{X,Y} \quad & \|A - Y\|_F^2 - \langle \Lambda, Y - \Delta \rangle + \frac{\mu}{2}\|Y - \Delta\|_F^2 \\ \text{s.t.} \quad & 0 \leq X \leq 1 \end{aligned} \tag{4}$$

Let the objective function be denoted as $L(X, Y; \Lambda; \mu)$ where Λ and μ are optimization parameters. Our proposed algorithm is given in Algorithm 1. Although there are several ways to compute (5) in Alg. 1, we adopt a simple gradient descent approach, where we calculate the gradients $\nabla L(X_k)$ and $\nabla L(Y_k)$ of L wrt X and Y respectively (see supplement). Then, the variable X is updated as $X_{k+1} = X_k - \alpha_k \nabla L(X_k)$, where α_k is the step size of the k^{th} iteration computed using a simple backtracking line search strategy. A similar update is used for Y. Momentum methods or Quasi-Newton methods can be used, if desired, but the simplicity of the gradient descent step provides computational advantage for larger datasets.

Algorithm 1 Augmented Lagrangian Method (ALM)

Given $X_0, Y_0 = X_0, \mu_0 > 0, \Lambda_0$. ($X_k$ denotes the value of X at the k-th iteration.)

while convergence not satisfied **do**

Compute an approximate minimizer of L i.e.

$$(X_{k+1}, Y_{k+1}) = \arg\min_{X,Y} L(\cdot, \cdot, \Lambda_k; \mu_k) \tag{5}$$

Project X_{k+1} to the feasible set i.e. $0 \leq X \leq 1$ and check for convergence (stopping criteria).

Set $\Lambda_{k+1} = \Lambda_{k+1} - \mu_k \Delta_k$

Update $\mu_{k+1} \geq \mu_k$. For instance, we can set $\mu_{k+1} = \beta\mu_k$ for a constant $\beta > 1$.

end while

In closing, we point out a few characteristics of the problem. First, because A has all values between 0 and 1, after a few iterations of ALM, the objective function is dominated by the penalty term with μ. Therefore, the problem behaves like a convex objective as the Hessian becomes diagonally dominant with a positive entry. Despite its simplicity, ALM performs quite well empirically (relative to existing methods) as we will discuss shortly. Next, we derive a Non-negative matrix factorization (NMF) [18] based model which offers other computational advantages.

2.2. NMF Formulation

The approach in [34] formulates the binary hashing problem as a factorization of the affinity matrix into symmetric components. It turns out that the Hamming distance matrix can also be expressed in such a factored form, though in this case, the factors generated will *not* be symmetric. This leads to a binary NMF type formulation for the problem, with additional constraints. The formulation offers several benefits which we will discuss shortly.

Our construction proceeds as follows. For notational purposes, let $\boldsymbol{A}$,$\boldsymbol{A_h}$, $\boldsymbol{X}$ be the corresponding vector forms of matrices A, A_h and X respectively. That is, if $A \in \mathbb{R}^{n\times n}$, then $\boldsymbol{A}$ is a column vector of length n^2, created from A in row major order. Clearly, (1) can be written as

$$\min_{A_h} \|\boldsymbol{A} - \boldsymbol{A_h}\|_F^2 \quad \text{s.t.} \quad \boldsymbol{A_h} = \Phi'(\boldsymbol{X}) \tag{6}$$

where $\Phi'(\cdot)$ serves as the Hamming distance function.

Next, we rewrite the matrix product $(1 - X)X^T$ in its vector form as well. This can be expressed as $H\boldsymbol{X}$, where $H = (I_{n\times n} \otimes (1 - X))$ is a $n^2 \times nd$ matrix where I denotes the identity matrix and $\otimes$ is the Kronecker product of two matrices (Chapter 5, [29]). To ensure that $\boldsymbol{A_h}$ is the vectorized form of a *symmetric* matrix, we multiply $H\boldsymbol{X}$, on the left, with a specific matrix M. Here, M is a constant binary matrix in $\mathbb{R}^{n^2\times n^2}$ constructed in the following way. Let i and j be two indices in $\boldsymbol{A}$ which correspond to symmetric off-diagonal entries in A: that is, $A(k, l) = \boldsymbol{A}(i)$ and $A(l, k) = \boldsymbol{A}(j)$. Then, the ith column of M, given as $M_{.i}$, contains 1s in rows i and j. The jth column is identical to the ith column, making M symmetric. Essentially, if $\boldsymbol{X}$ is a length appropriate vector, then $(M\boldsymbol{X})_i = \boldsymbol{X}_i + \boldsymbol{X}_j$, adds the corresponding off-diagonal elements. Then, it is easy to verify that $MH\boldsymbol{X}$ is the same as $\boldsymbol{A_h}$.

Since the constraint $\boldsymbol{X}_i(1 - \boldsymbol{X}_i) = 0$ is satisfied if and only if $\boldsymbol{X}_i \in \{0, 1\}$ and imposing the constraints as described above in (6), we get the following model,

$$\begin{aligned} \min_{\boldsymbol{X},H} \quad & \|\boldsymbol{A} - MH\boldsymbol{X}\|_F^2 \\ \text{s.t.} \quad & H = I \otimes (1 - \boldsymbol{X}), \quad \boldsymbol{X}_i \in \{0, 1\} \forall i. \end{aligned}$$

The formulation is similar to a binary NMF problem with additional constraints. Several approaches have been proposed recently to solve the NMF problem, where one or both factors are binary [32, 36]. For our purpose, we adopt a simple gradient projection scheme as discussed in [26]. Relaxing the binary restrictions, we define the convex relaxation of the feasible set as,

$$C := \{\{H, \boldsymbol{X}\} | H = I \otimes (1 - X), 0 \leq X \leq 1\} \tag{7}$$

We define the projection operator Π_C as,

$$\Pi_C(z) := \arg\min_{x\in C} \|x - z\|_p^p \tag{8}$$

$\Pi_C(z)$ exists and it is unique since C is convex. The choice of p does not matter but it is interesting to note that when $p = 1$, the projection operation [18] can be formulated as a linear program.

2.2.1 A Multiplicative Update scheme

Because (8) is in a NMF form, we can now derive solutions based on multiplicative updates.

Algorithm 2 Multiplicative Gradient Projection Algorithm

Start from an initial point $H, \boldsymbol{X} \in C$
while convergence not satisfied **do**
  Update $\boldsymbol{X}$ and H using the multiplicative update as given in (10) and (11).
  Update $(H, \boldsymbol{X}) \leftarrow \Pi_C(H, \boldsymbol{X})$.
end while

Let $f(H, \boldsymbol{X})$ be the objective function. The derivative of f with respect to X and H are given by,

$$\begin{aligned} \frac{\partial f}{\partial H} &= 2M\left(MH\boldsymbol{X}\boldsymbol{X}^T - \boldsymbol{A}\boldsymbol{X}^T\right), \\ \frac{\partial f}{\partial \boldsymbol{X}} &= 2H^T M(\ MH\boldsymbol{X} - \boldsymbol{A}). \end{aligned} \tag{9}$$

We now follow the recipe in [18] and perform an update as,

$$\boldsymbol{X}_l \leftarrow \boldsymbol{X}_l \frac{\left(H^T M\boldsymbol{A}\right)_l}{\left(H^T M^2 H\boldsymbol{X}\right)_l} \tag{10}$$

$$H_{ij} \leftarrow H_{ij} \frac{\left(MA\boldsymbol{X}^T\right)_{ij}}{\left(M^2 HXX^T\right)_{ij}}, \tag{11}$$

After this step, we project on to C to maintain feasibility.

ALM versus NMF. Occasionally, a NMF algorithm may fail to converge for a number of reasons. One can easily construct an example in one dimension to verify this. Numerical experiments in the later sections do indicate this nature of multiplicative updates in one specific case. The convergence proof (for multiplicative updates) is based on the construction of an auxiliary function. When the auxiliary function is non-smooth or when we are already at a stationary point of the auxiliary function, then the algorithm may fail to converge. We refer the reader to [18] for convergence details. Nonetheless, note that the update steps in principle require large size matrices, but due to the block structure of H and the construction of M, it can be broken down into much smaller computations, much of which can be done in parallel [22]. Multiplicative updates intuitively makes more sense when we are dealing with massive data sets since we need very limited information in order to update each entry. Second, this method is much more stable to incomplete data, by design. Third, NMF has been extensively studied in machine learning and mature implementations are available. The formulation makes the use of such libraries for binary hashing possible with minor cosmetic changes. Finally, ALM has a parameter μ which has to be carefully updated. This is because when we make $\mu^{k+1} \gg \mu^k$ then we try to satisfy the constraints more than seeking to decrease the objective.

2.2.2 Parallel Implementation

It turns out that because of the block structure of the matrix H, a number of key steps in the matrix updates can be simplified and done in a parallel manner. Consider the updates in (10) and (11). Since $\boldsymbol{A}$ is a vector of a symmetric matrix, $M\boldsymbol{A}$ is simply the vector form of $\hat{A} = 2A - \text{diag}(A)I$. We focus on the update rule for H first and observe that, if the H from a previous step is block diagonal (true for any feasible H), then the update equation will result in non-zero values only in the diagonal blocks of the new H matrix. Therefore, we can reformulate the rule only for the diagonal blocks of H (H_i) each of which is of size $n \times d$ as

$$H_i \leftarrow H_i \frac{\hat{A}(i,:)^T X(i,:)}{2(XH_i(i,:)^T X(i,:) + H_i X(i,:)^T X(i,:))} \tag{12}$$

Each of these block updates can be done in parallel since there is no dependency among the different blocks. The update rule for $\boldsymbol{X}$ can be written in terms of a matrix as

$$X \leftarrow X \frac{\hat{A}(i,:)^T H_i}{2(H_i(i,:)X^T H_i + X(i,:)H_i^T H_i)} \tag{13}$$

Once again this update rule can be further broken down to an update for each row of X, which can be done concurrently. We implemented this algorithm using CUDA, where computing the matrix products in the update rules is done concurrently at the level of individual entries of the matrix products. This gives a significant speedup compared to a non-parallel version, particularly when n grows larger. We will discuss this further in the experiments.

3. Initialization

The models outlined above are iterative approaches that need an initialization. To this end, we adopt a simple scheme, where we first project $\hat{X}$ to $d \ll D$ dimensions (either using PCA or random projections). Then, each point in the projected data is 'mean-thresholded' to get an initial estimate of X. Each real-valued entry is rounded to 0 if smaller than the mean of the corresponding dimension, and 1 otherwise (one can also obtain the threshold by computing the maximum margin in 1-D [28]). Consider the 'event' that our initialization procedure is poor in the sense that pairs of examples (in the real-valued distribution in $\mathbb{R}^d$) which are close are *not* thresholded to the same binary bit. We will bound (away from 1) the likelihood of such events under the assumption that the projected data follows some Gamma distribution (Chapter 19, [33]). This assumption allows the result to be general enough to include a wide variety of data distributions, yet provide useful initializations [2, 4].

Theorem 3.1 (Consistency of Initial estimate). *Let* $\{x_i\} \in \{0,1\}$ *be the binary embedding of* $\{v_i\} \sim \text{Gamma}(k,\theta)$ *where* $k \geq 1$ *(k, θ, kθ is shape, scale, mean*

of the distribution) so that

$$x_i = \begin{cases} 0 \;\; \textit{if} \;\; v_i \leq \mathbb{E}(v_i) = k\theta \\ 1 \;\; \textit{otherwise} \end{cases} \tag{14}$$

and v_i, v_j be any two entries (for examples i and j) in the same dimension $\in \{1, \cdots, d\}$ with $|v_i - v_j| = s < k\theta$. Then we have,

$$Pr(x_i \oplus x_j = 1) \leq f(s,k,\theta)g(k) \;\; \textit{where}$$
$$f(s,k,\theta) = \frac{\exp(\frac{-s}{\theta})}{\theta^2}\left(1 + \frac{s}{k\theta - s}\right)^{k-1}, \tag{15}$$
$$g(k) = \frac{\gamma(2k-1, 2k)}{2^{2k-1}[\Gamma(k)]^2}$$

Proof (Sketch). The proof (see supplement) follows by first observing that

$$x_i \oplus x_j = 1 \iff k\theta - s < v_j \leq k\theta$$

for some $v_j \leq v_i$. Hence, to show (15), we need to compute the following integral,

$$Pr(x_i \oplus x_j = 1) := \int_{v_j \in (k\theta - s, k\theta], v_i = v_j + s} p(v_i)p(v_j)$$

Using that fact that $v_i, v_j \sim \text{Gamma}(k, \theta), k \geq 1$ with $s < k\theta$, the above integral simplifies to

$$Pr(x_i \oplus x_j = 1) \leq \frac{\exp(-\frac{s}{\theta})(1 + \frac{s}{k\theta - s})^{k-1}}{(\Gamma(k))^2 \theta^2 2^{2k-1}} \bar{\gamma}(k)$$
$$\text{where } \bar{\gamma}(k) = \left[\gamma(2k-1, 2k) - \gamma\left(2k-1, \frac{2(k\theta - s)}{\theta}\right)\right]$$

where $\gamma(\cdot, \cdot)$ is the lower incomplete Gamma function. Observing that $\gamma(\cdot, \cdot)$ is positive and non-decreasing in its second argument, we get (15).

Whenever $k \geq 1$, $g(k)$ in (15) is smaller than 0.45 and decreases as k increases. Further, $f(s,k,\theta) < 1$, except for very small θ. The case of $\theta \approx 0$ can be rectified by rescaling the data to ensure θ is large enough. Overall, for some (k, θ), a decrease in s will drastically reduce the upper bound in (15). This makes sense since the binary embedding would contain very few errors at small values of s. Note that in our setting, it is not necessary to analyze the case of $s > k\theta$ since the instances i, j will be assigned different embeddings ($\{0, 1\}$) with probability 1 anyway. The above result is dimension-wise and applies directly to multiple dimensions (where each dimension has its own k and θ). The upper bound then will be the product of (15) type upper bounds over all dimensions.

Note that using the initialization as the starting point, Alg 1 and 2 monotonically decreases the objective. So the initialization serves as an upper bound on the final objective value. Hence, Thm 3.1 applies to the output of these algorithms as well.

3.1. Out of Sample Extensions

When evaluating on unseen data, we generate d linear classifiers, using the d bits of training code as labels, and then these classifiers are to predict the code for test example. Note that while most approaches in unsupervised hashing learn the seperating hyperplanes during the process of hashing itself, our method (similar to [20]) solves for the code first and then generates the hyperplanes, which are used to determine the codes for unseen test points. One advantage of our approach is that the hyperplanes generated are based on the maximum margin principle, which may result in more discriminative seperation of the two classes, for each hash bit, and can be especially useful for unseen data.

Time Complexity: For NMF approach, the update rules (12) and (13) have a time complexity of $O(n^2)$. However, in the parallel version, simultaneous updates reduce per iteration cost to $O(n)$. For ALM, update steps consist of matrix multiplications of the form XX^T which is $O(n^2)$ but much cheaper in practice. Using a standard package like Intel MKL reduces the run time to $O(n)$ by parallelization. The cost for out of sample extension is $O(dT)$ where T is the runtime for learning a linear classifier.

4. Experiments

We performed a number of experiments to evaluate the efficacy of our ALM and NMF methods, comparing these with five other approaches for binary hashing, including Locality Sensitive Hashing (LSH) [13], KLSH [17], SH [35], MDSH [34], BRE [16], ITQ [11] and Anchor Graph Hashing (1 and 2 layer) [24, 23]. We evaluate these on a number of machine learning and vision datasets, which vary in size, dimensionality, and number of classes. These include Iris, Heart, Nursery, MNIST, Caltech101, LabelMe, and two CIFAR datasets. All except LabelMe, come with class labels. Note that since ours is an unsupervised approach, labels are not used while learning the codes, but only for evaluation. For naming purposes, the data used to construct the distance matrix on which codes are learned is called *training* and all other query data are considered *test data*. For datasets where labels are available, we report accuracies of the k nearest Neighbors ($k = 4$) of a given query, w.r.t. the same class labels. This will demonstrate whether semantic concepts can be identified using such an approach.

We also evaluate the approaches by computing precision and recall values which are standard for such tasks, to see how well these approaches perform in approximating distances in the original space, both for training (next paragraph) and test data points.

Estimation of Distances: We use the Iris and Heart datasets to see how well the generated code approximates a given distance matrix. We use Euclidean distance for A, and the standard Gaussian kernels for affinity. For each

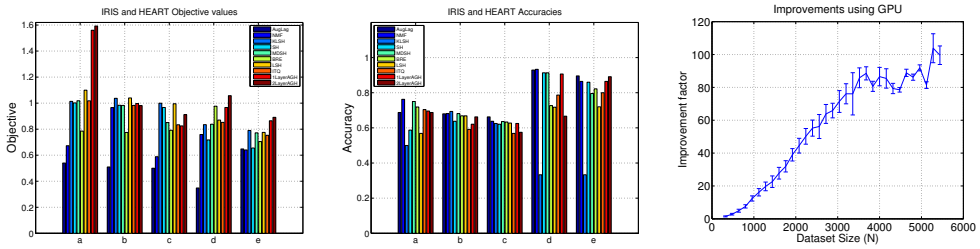

Figure 1. Objective function value (left) and NN accuracy (center) on Iris and Heart dataset. (a) Iris 2 bits, (b) Iris 4 bits, (c) Heart 4 bits, (d) Heart 8 bits, (e) Heart 16 bits. (right) Running time factor improvements for each iteration of the parallel implementation relative to the regular implementation

code, we compute $\|A - A_h\|^2$, and then normalize it by the squared Frobenius norm of A, ($\|A\|^2$) and the number of bits. Fig. 1 (columns 1-2) shows these values for both datasets, as a function of the number of bits. Note that we only choose bits which are powers of 2 and do not exceed the inherent dimension of the data. As can be seen from this plot, both our NMF and ALM approach approximate the distances better than any other method. By increasing the number of bits, the approximation progressively improves for all approaches.

Number of classes: Caltech101 is a good dataset to see how an increase in number of classes affect performance, since it has a large number of classes (up to 101) and a small number of data items per class, which makes it a difficult dataset in some sense. Since this is widely used in computer vision, kernels for this dataset on different features are available (UCSD MKL dataset). We use the (mean of a small set of) kernel(s) as the input to some of our comparison methods and also to create our distances. However, since the original features for the test dataset (needed by some of the methods being compared to) is unavailable, we limit our evaluation to the training dataset only. Whenever features were needed by our method or comparison methods for training data, we generated them by implementing the approach in [1]. Fig. 2 (Row 1, Col 1-2) shows the results on the Caltech 101 dataset, using 50 and 101 classes. The plot shows that in both settings, our methods outperform the other approaches. In addition, doubling the number of classes affects performance but not drastically.

Dimensionality: We use Cifar (Cifar-10), Cifar-100 and MNIST to see how our methods behave compared to others on really high dimensional datasets (Fig. 2). HOG features extracted from the Cifar data is a feature vector of length 625 for each image, whereas MNIST's native dimension is 784. For MNIST (Row 1, Col 3 and Row 2, Col 1), our methods and SH are the top performers, whereas in the Cifar case(Row 2, Col 2), our algorithms give the best results for test data. For Cifar-100 (Row 2, Col 3), the ALM method outperforms NMF and others, particularly in testing. More results on these datasets are in the supplement.

Generalization to unseen data as a function of distances: Here, we evaluate how well distances are approximated with all the datasets mentioned earlier. However, due to lack of space, we only present a subset of these plots here. In order to evaluate how well NN distances are estimated using codes on unseen data, we first define a threshold, such that if the Euclidean distance of two points is less than the threshold, they are considered "true neighbors". Given a query and a threshold on Hamming distance (in our case, we set it to 3), the retrieved items for the query are all datapoints whose Hamming distance is below the threshold. We compute *precision* as the proportion of retrieved points that are indeed true neighbors, and *recall* as the proportion of true retrieved points out of all true points. For LabelMe, the size of training dataset is set at 2000 and the remaining examples (size 3000) are used for testing. Fig. 2 (Row 4, Col 1) shows the precision values as a function of number of bits. In general, we outperform all other methods w.r.t. this measure. Fig. 2 (Row 4, Col 2-3) shows precision as a function of recall for this setting for bits 4 and 16. Here, we see that our performance is better than other methods in both setting. The precision results on two other datasets Mnist (Row 3, Col 2) and Nursery (Row 3, Col 3) show similar results. (Note that the precision and PR curves for all the other datasets are included in the supplement).

Generalization to unseen data as a function of labels: For 4 datasets, Nursery (Row 3, Col 1), MNIST, Cifar and Cifar-100, we show plots (Fig. 2) where the codes are learnt on a training size of 2000 and generalized to a much larger testing set ($\geq$ 10000 in some cases). While the relative percentage varies (depending on the dataset), in most cases, our models show impressive performance in finding neighbors which have the same class label as the query point. Furthermore, we also performed experiments to see if increasing the size of the training set improves generalization but found that in general, the relative improvement saturates pretty quickly. Therefore, a moderate sized training set (if chosen randomly) ensures good generalization.

Running Time: Note that when comparing all methods (including our nonparallel methods) with respect to run time, the performances are not significantly different. But we do see a significant difference when comparing (any) nonparallel method with the parallel implementation. To do this, we implemented our Parallel algorithm using Matlab's

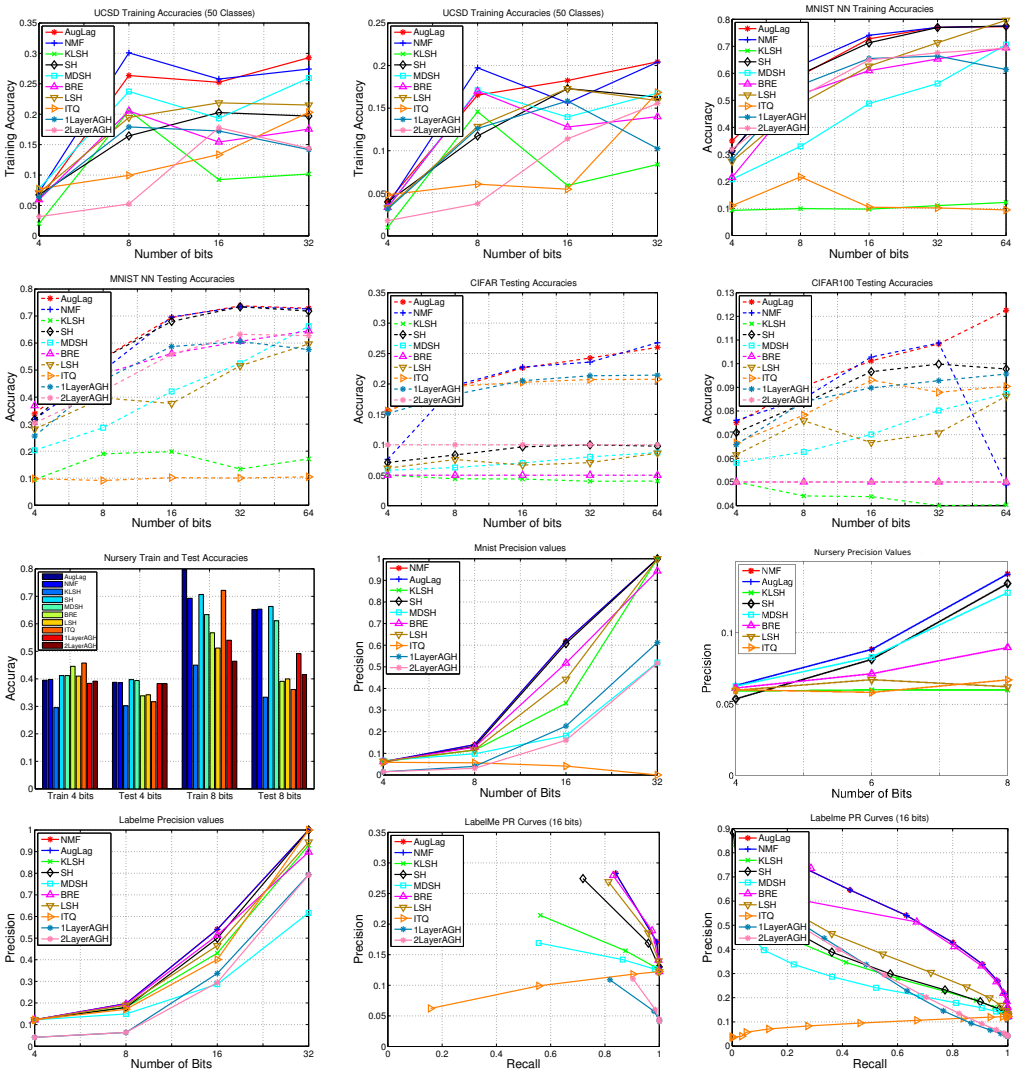

Figure 2. Results on UCSD (Row 1, Cols 1-2), MNIST (Row 1, Cols 3; Row 2 Col 1), Cifar and Cifar100 (Row 2, Cols 2-3), Nursery (Row 3, Col 1), Precision curves for Mnist (Row 3, Col 2) and Nursery (Row 3, Col 3) and Labelme (Row 4) datasets.

parallel computing toolbox and CUDA 5.5 on a machine with a Tesla K40 graphics card. We measure the average time taken per iteration using the GPU implementation as well as our version which does not use GPU. We evaluate the factor improvement in runtime when a GPU implementation is used. The results are plotted in Fig. 1 (column 3). Overall, we get 100x improvement in the runtime as we increase n after which the gains saturate, perhaps due to the number of threads supported on the device [9]. For $n = 1000$ the runtime is < 0.2s. These improvements are significant compared to a non-parallel version, resulting in iteration times in the order of milliseconds.

5. Conclusions

This paper proposes effective solutions to the binary hashing problem where, the goal is to generate binary codes for high dimensional data points such that the Hamming distance maintains high fidelity with a target distance function. We first derive an Augmented Lagrangian method as a reference - later, we provide a NMF based model which leads to a Multiplicative updates scheme for binary hashing. The procedure is parallelizable, by design, and will enable deploying hashing for very large scale datasets using cosmetic changes to heavily engineered implementations that are already widely available (e.g., MapReduce [6]). We show via extensive experiments that both ALM and NMF methods yield impressive performance relative to existing methods.

Acknowledgments: The authors were supported via grants NSF RI 1116584, NSF CGV 1219016 and NSF CAREER award 1252725. The research was also partial supported by University of Wisconsin CPCP (U54 AI117924) and University of Wisconsin ICTR (UL1TR000427).

References

[1] M. Balcan, A. Blum, and S. Vempala. Kernels as features: On kernels, margins, and low-dimensional mappings. *Machine Learning*, 65(1):79–94, 2006.

[2] K. Bowman and L. Shenton. *Properties of estimators for the gamma distribution*. Marcel Dekker New York, 1988.

[3] J. Chuang, S. Gupta, C. Manning, and J. Heer. Topic model diagnostics: Assessing domain relevance via topical alignment. In *ICML*, 2013.

[4] D. Clark and C. Thayer. A primer on the exponential family of distributions. In *Casualty Actuarial Society Spring Forum*, pages 117–148, 2004.

[5] S. Dasgupta. Experiments with random projection. *CoRR*, abs/1301.3849, 2013.

[6] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] A. Del Bue, J. Xavier, L. Agapito, and M. Paladini. Bilinear factorization via augmented lagrange multipliers. In *ECCV*, pages 283–296. 2010.

[8] W. Dong, M. Charikar, and K. Li. Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces. In *SIGIR*, 2008.

[9] K. Fatahalian, J. Sugerman, and P. Hanrahan. Understanding the efficiency of gpu algorithms for matrix-matrix multiplication. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS*, 2004.

[10] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011.

[11] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *PAMI*, 35(12):2916–2929, 2013.

[12] K. Grauman and R. Fergus. Learning binary hash codes for large-scale image search. In *Machine Learning for Computer Vision*, pages 49–87. Springer, 2013.

[13] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, 1998.

[14] Y. Ke, D. Hoiem, and R. Sukthankar. Computer vision for music identification. In *CVPR*, 2005.

[15] Y. Ke, R. Sukthankar, and L. Huston. An efficient parts-based near-duplicate and sub-image retrieval system. In *ICM*, 2004.

[16] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, 2009.

[17] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2009.

[18] D. Lee and H. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, 2000.

[19] H. Li, W. Liu, and H. Ji. Two-stage hashing for fast document retrieval. In *ACL*, 2014.

[20] G. Lin, C. Shen, D. Suter, and A. van den Hengel. A general two-step approach to learning-based hashing. In *ICCV*, 2013.

[21] R. Lin, D. Ross, and J. Yagnik. Spec hashing: Similarity preserving algorithm for entropy-based coding. In *CVPR*, 2010.

[22] C. Liu, H. Yang, J. Fan, L. He, and Y. Wang. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce. In *WWW*, 2010.

[23] W. Liu, C. Mu, S. Kumar, and S. Chang. Discrete graph hashing. In *NIPS*, 2014.

[24] W. Liu, J. Wang, S. Kumar, and S. Chang. Hashing with graphs. In *ICML*, 2011.

[25] B. Neyshabur, N. Srebro, R. Salakhutdinov, Y. Makarychev, and P. Yadollahpour. The power of asymmetry in binary hashing. In *NIPS*, 2013.

[26] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[27] M. Norouzi and D. Blei. Minimal loss hashing for compact binary codes. In *ICML*, 2011.

[28] J. Peng, L. Mukherjee, V. Singh, D. Schuurmans, and L. Xu. An efficient algorithm for maximal margin clustering. *JGO*, 52(1), 2012.

[29] K. Petersen and M. Pedersen. The matrix cookbook. *Technical University of Denmark*, 2006.

[30] R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[31] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. In *ICCV*, 2003.

[32] M. Slawski, M. Hein, and P. Lutsik. Matrix factorization with binary components. In *NIPS*, 2013.

[33] C. Walck. Handbook on statistical distributions for experimentalists, 2007.

[34] Y. Weiss, R. Fergus, and A. Torralba. Multidimensional spectral hashing. In *ECCV*, 2012.

[35] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2008.

[36] Z. Zhang, C. Ding, T. Li, and X. Zhang. Binary matrix factorization with applications. In *ICDM*, 2007.