

Multidimensional Spectral Hashing

Yair Weiss, Rob Fergus, and Antonio Torralba

School of Computer Science, Dept. of Computer Science, CSAIL,
Hebrew University Courant Institute, MIT
New York University
yweiss@cs.huji.ac.il, fergus@cs.nyu.edu, torralba@csail.mit.edu

Abstract. With the growing availability of very large image databases, there has been a surge of interest in methods based on “semantic hashing”, i.e. compact binary codes of data-points so that the Hamming distance between codewords correlates with similarity. In reviewing and comparing existing methods, we show that their relative performance can change drastically depending on the definition of ground-truth neighbors. Motivated by this finding, we propose a new formulation for learning binary codes which seeks to reconstruct the *affinity* between datapoints, rather than their distances. We show that this criterion is intractable to solve exactly, but a spectral relaxation gives an algorithm where the bits correspond to thresholded eigenvectors of the affinity matrix, and as the number of datapoints goes to infinity these eigenvectors converge to eigenfunctions of Laplace-Beltrami operators, similar to the recently proposed Spectral Hashing (SH) method. Unlike SH whose performance may degrade as the number of bits increases, the optimal code using our formulation is guaranteed to faithfully reproduce the affinities as the number of bits increases. We show that the number of eigenfunctions needed may increase exponentially with dimension, but introduce a “kernel trick” to allow us to compute with an exponentially large number of bits but using only memory and computation that grows linearly with dimension. Experiments shows that MDSH outperforms the state-of-the-art, especially in the challenging regime of small distance thresholds.

1 Introduction

Perhaps the most salient aspect of computer vision and machine learning in recent years is the widespread availability of gigantic datasets with millions or trillions of items. This effect, sometimes called the “data deluge” creates a tremendous opportunity for machine learning applied to vision but also requires new algorithms that can efficiently work with such large datasets, where even a single, linear scan through the data may be prohibitively expensive.

In theoretical CS, sublinear time search methods are achieved through techniques called Locality Sensitive Hashing (LSH)[1]. In the simplest version of LSH, called E2-LSH [2], the i th bit in the code of an item x is simply given by $sign(w_i^T x - b_i)$, where w_i is randomly chosen vector and b_i a randomly chosen threshold. As the number of bits goes to infinity, it can be shown that Hamming

distance in an E2-LSH code will be a monotonic function of the original Euclidean distance. In [3] it was shown how to construct LSH codes that provably approximate a kernel function of the distance, rather than the raw distance.

Despite the theoretical guarantees of such randomized approaches in the limit of many bits, the performance with a limited bit budget is often unsatisfactory, and this has led to interest in approaches that *learn* the hash codes from data. Semantic Hashing proposed by Salukadnitov and Hinton[4] trains a multi-layer neural network to learn the binary code. Codes trained in this fashion can outperform random codes in both text and image applications[5].

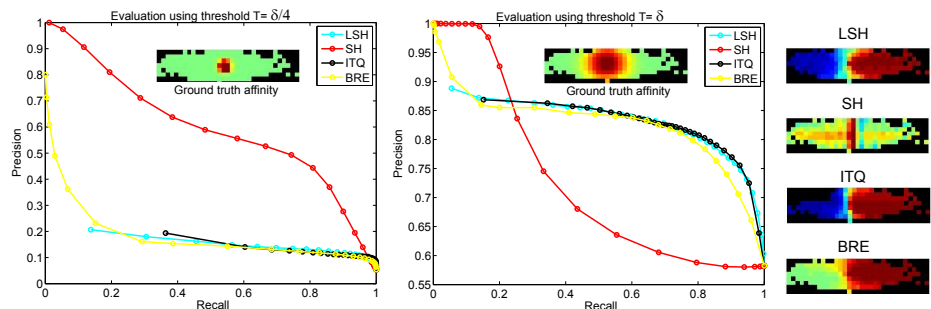


Fig. 1. The puzzle of previous work. The exact same algorithms with the exact same data (32 dimensional Gaussian) give a very different ranking of performance depending on our definition of ground truth neighbors. When the neighborhood for the ground truth is small (left), SH outperforms all other algorithms, and when it is large (right) it performs the worst. The rightmost images show the Hamming similarities of the different codes, projected down to 2D.

Due to the complexity of training the multilayer neural net, in recent years there has been much interest in other training algorithms for short binary codes. Weiss et al.[6] formulated the problem as an optimization problem and searched for bits that are independent and also minimize the expected Hamming distance between similar datapoints. They showed that this criterion is intractable but a spectral relaxation leads to codes where each bit corresponds to thresholds of eigenfunctions of the Laplace Beltrami operator, and suggested approximating the distribution of the data with simple parametric families. Xu et al.[7] optimize a similar criterion but show how to use multiple hash tables to speed up retrieval significantly. Kumar et al.[8] proposed an alternative criterion where one seeks to maximize the variance of the bits, and showed that this criterion leads to a code where each bit is sign of a PCA component. Kulis and Darrell[9] suggested optimizing the L2 loss between the Hamming distances and the original Euclidean distances. They showed that optimizing one bit given all the others can be done in closed form. In [10] a hinge loss rather than L2 loss is used, and it is optimized using techniques from structured prediction. Lin et al. [11] suggested an algorithm that incrementally adds bits to increase the Hamming distance between dissimilar objects while keeping the Hamming distance between similar

objects small. Yao and Lazebnik[12] also suggested looking at the difference between Hamming distances and Euclidean distances, and suggested an algorithm that finds a rotation of the PCA vectors so that after rotation the thresholding operation causes less quantization errors. They also showed that the sign of a random rotation of the PCA coefficients often performs better than the sign the original PCA coefficients.

Given that the recent approaches to learn binary codes use different optimization criteria, it is not surprising that they perform differently on different datasets. What is surprising, however, is that when performance of the algorithms is evaluated using standard precision-recall curves, the ranking of the algorithms can change dramatically even on the same dataset. In order to use precision-recall curves, one first defines a threshold on the original Euclidean distance. Two datapoints are defined as similar if their Euclidean distance is less than the threshold. Given a query and a threshold on Hamming distance, the retrieved items for the query are all datapoints whose Hamming distance is below the threshold. Precision measures the proportion of retrieved points who are indeed similar, while recall measures the proportion of similar retrieved points out of all similar points.

To illustrate the different ranking of the algorithms, consider Figure 1 where we show precision-recall curves¹ for Spectral Hashing [6] (SH), LSH [2], Gong and Lazebnik’s ITQ [12] and Kulis and Darrell’s BRE [9]. The data are sampled from a 32 dimensional Gaussian, where the standard deviation for the d th dimension is given by $(1/d)^2$ and all codes are 32 bits. *In both figures, the training data, test data and algorithm results are identical. The only difference is the threshold that defines a similar neighbor.* In the right column this threshold T is set to be δ , the average distance between any two points in the training set, while in the left column it is set to be $\delta/4$. It can be seen that this threshold makes a huge difference on the ranking of the algorithms. In particular, for the smaller neighborhoods, SH greatly outperforms the other techniques, while for the larger neighborhood it is the worst algorithm.

We can understand this difference in behavior by visualizing the Hamming distances from a query point to all other points in the dataset. Although the data is 32 dimensional, we visualize it in two dimensions using the first two PCA components (high values are high dot product between the binary code of a query point and a test point). It can be seen that Spectral Hashing does a very bad job of approximating the distance to far away points and collapses all of them to a similar distance, while the other methods do much better at capturing the far distances but not small distances.

Which of the algorithms shown in Figure 1 is therefore better? Obviously the answer is application dependent, as it depends on the scale of Euclidean distances that we want our algorithm to recover. Ideally, we would want an algorithm that learns different codes depending on the desired scale of distances. In this

¹ Note that since the Hamming distances are discrete, the precision recall curves are also discrete. We connect the points with lines for ease of visualization, but intermediate points on the lines are not achievable by the algorithms.

paper, we present such an algorithm. Our algorithm seeks a binary code so that inner product among the codes approximates the *affinity* between datapoints, i.e. $W(i, j) = \exp(-\|x_i - x_j\|^2/2\sigma^2)$. Since the affinity is a nonlinear function of distance, approximating the affinity uniformly will lead to approximating the distances with nonuniform quality. We show that approximating the affinity matrix can be formalized as a binary matrix factorization problem which is NP hard, but a simple relaxation yields a problem whose solution is simply given by the eigenvectors of the affinity matrix. A simple way to learn a binary code is therefore to simply threshold these eigenvectors, but we discuss two problems with this simple approach: (1) out-of-sample extension (i.e. calculating the code for a new datapoint) and (2) the curse of dimensionality whereby the number of eigenvectors needed to achieve good performance grows exponentially with dimension. By making a separability assumption on the data distribution and taking the limit of the eigenvectors as the number of datapoints goes to infinity we can avoid these two problems. In particular, we introduce a “kernel trick” that allows us to compute the factorization with an exponentially large number of eigenvectors, but using only time and memory that grows linearly with dimension. We compare our algorithm to the state-of-the-art on real and synthetic datasets.

2 A New Formulation for Learning Binary Codes

Typically the cost function for learning binary codes is given in terms of the Hamming distance $\|y_i - y_j\|$ between the binary codes of datapoints i and j , where y_i is a binary vector of length k . But since the elements of y_i are constrained to be plus or minus 1, the Hamming distance is a simple function of the dot product, or Hamming affinity $y_i^T y_j$, since: $\|y_i - y_j\|^2 = 2k - 2y_i^T y_j$. Thus instead of requiring that Hamming distances match some target value, we can equivalently require that the Hamming affinity match some target value. A natural choice for this target value is the *affinity* $W(i, j) = \exp(-\|x_i - x_j\|^2/2\sigma^2)$, where as noted above, the scale parameter σ is an explicit input set by the user. Another small modification we make to the standard formulation of learning binary codes, is that we allow a weight λ , one for each bit. The code for each datapoint is still binary valued (requiring very little memory and computation for each codeword), but the weights can be real-valued. Using these weights, the Hamming affinity is a linear combination of the agreement between the codes: $y_i^T A y_j = \sum_k \lambda_k y_i(k) y_j(k)$, where A is a diagonal matrix whose diagonal values give the weights. We seek codes such that this weighted Hamming affinity is equal to the original affinity $y_i^T A y_j \approx W(i, j)$:

$$(Y^*, A^*) = \arg \min_{y_i \in \{-1, 1\}^k, A} \sum_{i, j} (y_i^T A y_j - W(i, j))^2 \quad (1)$$

Let Y be a $n \times k$ matrix, where the i th row gives the binary code of the i th datapoint, then the cost function can be rewritten as minimizing the Frobenius

norm between the affinity matrix W and the rank k matrix YAY^T :

$$(Y^*, A^*) = \arg \min_{y_i \in \{-1, 1\}^k, A} \|W - YAY^T\|_F^2 \quad (2)$$

Under this formalization, the best binary code for a given dataset is equivalent to performing *binary matrix factorization* of the affinity matrix.

Before discussing how to optimize equation 2, we point out some of properties of this formulation and contrast it with related approaches.

- As the number of bits approaches infinity, the optimal code using our formulation (eq. 2) is guaranteed to faithfully reproduce the affinity W . In other words, with an infinite number of bits, the Hamming affinity will be inversely related to Euclidean distance for all pairs of points. This should be contrasted with approaches such as SH [6] whose performance may actually deteriorate as the number of bits increases [3].
- For a finite bit budget, the optimal code using our formulation will give different approximations to the distance depending on a user-supplied parameter σ . Since the affinity maps all distances much greater than σ to a number that approaches zero, the optimal code will not need to exactly reproduce the distances between far-away pairs of points, and can simply give an affinity that is close to zero for all pairs of distant points. This should be contrasted with approaches such as ITQ, BRE and LSH that attempt to faithfully reproduce all Euclidean distances. Since the number of pairs of points that are distant far outweigh the pairs that are nearby, approaches that try to reproduce all distances will often fail to reproduce the small distances when given a limited bit budget (see figure 1).
- The optimal code using our approach is not explicitly constructed so that the bits are independent. We do not require this since for a fixed bit budget, by seeking a low reconstruction error we are implicitly penalizing codes where one bit is redundant with other bits. This should be contrasted with approaches such as SH and USPLH[8], where independence between the bits is explicitly required.
- The mapping from a datapoint x_i to its code y_i is not restricted to any particular functional form. This is in contrast to almost all previous approaches (ITQ, BRE, LSH, MLH etc.) which restrict each bit to be the sign of $w^T x_i$ or (when using kernels) sign of $w^T f(x_i)$ for a fixed f .

Unfortunately, as in many constrained matrix factorization problems[13], the discrete constraint that the elements of the matrix are in $\{1, -1\}$ make this problem computationally intractable.

2.1 Spectral Relaxation

We notice that removing the binary constraint yields a standard matrix factorization problem for which the optimal solution is easily obtained by taking the top k eigenvectors of W . Denote by U a $n \times k$ matrix whose columns are the top k eigenvectors and by Λ a diagonal matrix with the k eigenvalues of W then $U\Lambda U^T$

is the best rank k approximation of W . If we are lucky, and the eigenvectors are already binary valued, then we have solved the binary matrix factorization problem. As we now discuss, even if the eigenvectors are not binary-valued they (1) give an upper bound on the performance of any code and (2) can be thresholded to give an approximate solution for binary matrix factorization.

The preceding discussion suggests an extremely simple method for learning binary codes that focus on a range of distances defined by σ — use σ to define an affinity matrix and threshold the eigenvectors of affinity matrix. But this approach will not give a practical code. In the case of the state-of-the-art algorithms (SH, LSH, ITQ) there is a simple function that allows us to calculate the code of a new datapoint x (in LSH and ITQ we simply perform k dot products of x with known vectors and threshold the result). But to calculate the binary codes using eigenvectors of the affinity matrix, we need to compute the affinity between x and all other datapoints, form the $(n+1) \times (n+1)$ affinity matrix and calculate its eigenvectors. Alternatively, we can use methods such as the Nystrom approximation [14] to approximate the eigenvectors of the affinity matrix from a smaller matrix, but this approach still requires prohibitive computation for large datasets. We now show how to calculate the limit of the eigenvectors as the number of points approaches infinity and how this leads to a practical, yet inefficient algorithm.

2.2 The limit of matrix factorization

To analyze the limit of the eigenvectors, we need to take a slightly different viewpoint (cf. [15, 6, 16–18]): rather than thinking of the n dimensional space defined by the n points, we consider the d dimensional space R^d in which these points lie. We define an operator $\mathbf{W}(s, t) = \exp(-\frac{\|s-t\|^2}{2\sigma^2})$ which simply measures the affinity between any two points s and t in R^d (unlike $W(i, j)$ which only measures affinity between points in the dataset). It is easy to see that when the points in the dataset are IID samples from a distribution $\Pr(x)$, the matrix factorization problem approaches a different problem, which we will call the operator factorization problem.

Claim 1: Let x_i be IID samples from $\Pr(x)$, and $W(i, j)$ the affinity between x_i and x_j . For any function $y(x)$, define $y_i = y(x_i)$ as the code corresponding to x_i , then as the number of samples goes to infinity $\frac{1}{N^2} \|W - Y^T AY\|_F$ approaches:

$$J(y, A) = \int_{s,t} (\mathbf{W}(s, t) - y^T(s)Ay(s))^2 \Pr(s) \Pr(t) ds dt \quad (3)$$

The proof follows directly from the law of large numbers.

In the operator factorization problem, we seek a code $y(s)$ so that for *any* two points s and t the affinity between the codes $y(t)^T Ay(s)$ is equal to the original affinity $W(s, t)$. The probability $\Pr(x)$ defines a weighting on the points, giving more weight to pairs of points (s, t) with higher density.

Just as the solution to the matrix factorization problem is given by the k eigenvectors of W , the solution to the operator factorization can be shown

to be given by k *eigenfunctions* of the operator \mathbf{WP} . We define the operator $(\mathbf{W}f)(s) = \int_t W(s,t)f(t)dt$ and the operator $(\mathbf{P}f)(s) = \Pr(s)f(s)$. A function f is an eigenfunction of \mathbf{WP} if $\mathbf{WP}f = \lambda f$.

Claim 2: The solution to the operator factorization problem, $(y^*, \Lambda^*) = \arg \min J(y, \Lambda)$, with J as in equation 3 and $y(s)$ constrained to be a k dimensional vector, is given by the k eigenfunctions of WP with maximal eigenvalue.

Proof: For the case that $\Pr(x)$ is uniform on a region in R^d , the result follows from the standard matrix factorization result. When $\Pr(x)$ is non-uniform, a change of variables $z = P^{1/2}y$ converts the problem into one with uniform $\Pr(x)$ and the operator replaced by $\sqrt{\mathbf{P}}\mathbf{W}\sqrt{\mathbf{P}}$. This means that the optimal z is given by the top k eigenfunctions of $\sqrt{\mathbf{P}}\mathbf{W}\sqrt{\mathbf{P}}$ or alternatively $\mathbf{WP}y = \Lambda y$.

Taken together, claims 1-2 suggest a solution to the binary matrix factorization problem: we simply solve for the top k eigenfunctions of the operator \mathbf{WP} , i.e. we solve for both the eigenfunction $\Phi_i(s)$, and their eigenvalues λ_i . Given these solutions, we can compute the code for each point x simply by thresholding at zero $y(x) = \{\text{sign}(\Phi_i(s))\}_{i=1}^k$. Note that this gives an analytical formula for the code of a new datapoint, so this algorithm is practical. Also, when the eigenfunctions happen to be binary valued, this code is optimal in the limit of many datapoints — no binary code can do a better job of approximating the affinity. However, this code may still be very inefficient due to the curse of dimensionality.

2.3 Curse of Dimensionality

How many bits will we need in order to faithfully approximate the original affinity? We can answer this question by analyzing the eigenspectrum of (\mathbf{WP}) . If there are only a small number of eigenvalues that dominate the spectrum, we can expect a good approximation with a small code. But if the number of large eigenvalues is very large, we cannot expect any small code to give a good approximation.

To illustrate this, consider the multidimensional uniform distribution that was analyzed in [6]. For this case the eigenfunctions of \mathbf{WP} are simply the eigenfunctions of \mathbf{W} (since P is uniform) and correspond to the fundamental modes of vibration of a metallic plate.² The eigenfunctions are all sinusoidal functions and can be divided into “single-dimension” eigenfunctions and “outer-product” eigenfunctions. All single-dimension eigenfunctions are of the form:

$$\Phi_j(x(i)) = \sin\left(\frac{\pi}{2} + \frac{j\pi}{b_i - a_i}x(i)\right) \quad (4)$$

$$\lambda_j = e^{-\frac{\sigma^2}{2} \left| \frac{j\pi}{b_i - a_i} \right|^2} \quad (5)$$

where $x(i)$ refers to the i th coordinate of x and $x(i)$ is uniformly distributed in $[a_i, b_i]$.

² In [6] these eigenfunctions arose from relaxing the balanced-cut problem, while here the same eigenfunctions arise from binary matrix factorization. For a uniform distribution, both problems give the same eigenfunctions with the eigenvalue λ replaced by $1 - \lambda$.

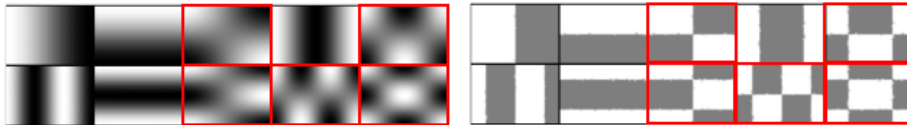


Fig. 2. Left: Eigenfunctions for a uniform rectangular distribution in 2D. **Right:** Binary codes obtained by thresholding these eigenfunctions. These eigenfunctions can be divided into single-dimension eigenfunctions and “outer-product” eigenfunctions that are indicated by a red-frame. In the standard SH algorithm, bits from the outer-product eigenfunctions are ignored while here we show how to efficiently compute Hamming distances with an exponential number of such bits using a “kernel trick”.

The “outer-product” eigenfunctions are simply products of eigenfunctions along different dimensions and their eigenvalue is simply the product of the eigenvalues along each dimension. Figure 2 shows the eigenfunctions for a uniform rectangular distribution in R^2 (replotted from [6]), with the “outer-product” eigenfunctions shown with a red frame.

In the ordinary spectral hashing (SH) algorithm, these “outer-product” eigenfunctions were discarded since the goal of SH is to search for codes where the bits are independent, and the outer-product eigenfunctions lead to bits that are deterministic functions of the other bits. But if our goal is to approximate the affinity matrix, then discarding these eigenfunctions makes no sense — in fact, the optimal factorization will often contain a very large number of these outer-product eigenfunctions.

In particular, when the uniform distribution is not over two dimensions, as in figure 2 but rather over d dimensions, then the number of outer-product eigenfunctions with large eigenvalue will grow exponentially with the dimension d . For example, if $\sigma \ll |b_i - a_i|$ for all i then the each dimension will have at least two eigenfunctions with the maximal eigenvalue $\lambda \approx 1$. In this case, *the number of bits needed to faithfully approximate the affinity grows exponentially with dimension*. Since the eigenfunctions give the *optimal* low-rank factorization of the affinity, this means that any code (binary or continuous) that does a good job of approximating the affinity should have a length that grows exponentially with the dimension.

Although we have focused above on the uniform distribution, it can be shown that these “outer-product” eigenfunctions, will exist whenever the distribution $\Pr(x)$ is a product of distributions along single dimensions. The bad news is that this means we will require an exponentially large number of bits to faithfully approximate the affinity. The good news, however, is that we do not need to store the bits corresponding to outer-product eigenfunctions, since we can calculate their values directly from the bits of the single-product eigenfunctions. This insight leads to the following algorithm, which we call *inefficient multidimensional spectral hashing*:

1. calculate the single-dimension eigenfunctions. We denote by $\Phi_{ij}(x(i))$ the j th eigenfunction of the i th coordinate and λ_{ij} the corresponding eigenvalue.
2. Sort λ_{ij} and find a set of k indices (I, J) so that λ_{ij} is maximal.

3. encode each datapoint x with $y(x) = \text{sign}(\phi_{ij}(x))$ for all $(i, j) \in (I, J)$.
4. Expand the code of x to include all outer-product eigenfunctions. For any subset $(I_l, J_l) \subset (I, J)$ such that no index i appears in I_l more than once, add an additional bit given by $y_l(x) = \prod_{i,j \in (I_l, J_l)} \text{sign}(\Phi_{i,j}(x(i)))$. The weight of this bit is given by $\lambda_l = \prod_{i,j \in (I_l, J_l)} \lambda_{i,j}$.
5. The Hamming affinity between x_i and x_j is given by:

$$H(i, j) = \sum_l y_l(x_i) \lambda_l y_l(x_j) \quad (6)$$

where the index l goes over the single-dimension bits as well as the outer-product bits.

We call this algorithm “multidimensional spectral hashing” (MDSH) because the first three steps (calculating the single-dimension eigenfunctions, sorting the eigenvalues and encoding using the sign of the eigenfunctions) are exactly the same as the original SH algorithm. But the difference is that whereas the original SH used only single-dimension eigenfunctions during retrieval, here we expand the code to include the outer-product eigenfunctions as well. From the standpoint of approximating the affinity, this makes a huge difference. Suppose instead of thresholding the eigenfunctions (in the third step of the algorithm) we would encode each point with the value of the k top single-dimension eigenfunctions. It is easy to show that in this case the approximate affinity calculated by the algorithm $H(i, j)$, will be better than any other factorization that uses continuous codes of length k_2 , where k_2 is the number of eigenfunctions (single dimension and outer-product) with eigenvalues greater than λ_k . In general k_2 will be much larger than k (it will grow exponentially in dimension) but the MDSH algorithm guarantees that we will obtain a factorization that is optimal with rank k_2 even though we only encode each point with a code of length k .

As its name suggests, the inefficient MDSH algorithm requires computation that scales exponentially with dimension (equation 6). However, we now show how to compute the same Hamming affinity with computation that is linear in dimension.

Observation: (Kernel Trick) Let $H(i, j)$ be the weighted Hamming affinity between two points, using all the bits in the inefficient MDSH algorithm (equation 6). Let $H_d(i, j)$ be the weighted Hamming affinity between these two points using only pure eigenfunctions along dimension d , i.e. $H_d(i, j) = \sum_{(d,l) \in (I,J)} \lambda_{dl} \text{sign}(\phi_{dl}(x_i(d))) \text{sign}(\phi_{dl}(x_j(d)))$. Then:

$$H(i, j) = -1 + \prod_d (1 + H_d(i, j)) \quad (7)$$

Proof: This follows directly by expanding the product in equation 7.

Combining the “kernel trick” with the inefficient MDSH algorithm gives our final algorithm, **Multi-Dimensional Spectral Hashing** (MDSH):

1. calculate the single-dimension eigenfunctions. We denote by $\Phi_{ij}(x(i))$ the j th eigenfunction of the i th coordinate and λ_{ij} the corresponding eigenvalue.
2. Sort λ_{ij} and find a set of k indices (I, J) so that λ_{ij} is maximal.

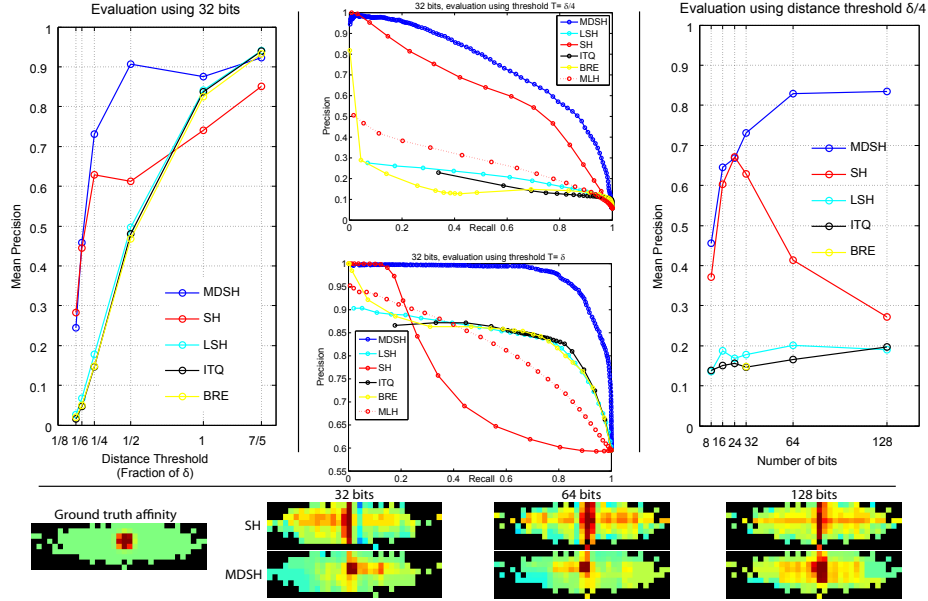


Fig. 3. Comparison between different algorithms on toy data (32D Gaussian from figure 1). We compare MDSH, SH [6], LSH [2], ITQ [12], BRE [9] and MLH [10]. **Left:** Performance as a function of distance threshold. Ranking of existing algorithm depends on distance, but MDSH consistently works best. **Middle:** Precision-recall curves for two different thresholds. **Right:** Performance as a function of number of bits. SH performance can decrease with more bits, but MDSH continues to improve. **Bottom:** A visualization of the affinity for SH and MDSH as the number of bits increases. With 128 bits, the SH affinity has spurious structure, while MDSH is close to the ground truth.

3. encode each datapoint x with $y(x) = \text{sign}(\phi_{ij}(x))$ for all $(i, j) \in (I, J)$.
4. The Hamming affinity between x_i and x_j is given by equation 7.

Note that this algorithm will give exactly the same Hamming similarity as the inefficient one but both storage and computation grow linearly with dimension. Instead of explicitly expanding the codes to include all the outer-product bits, the kernel trick allows us to compute *exactly* the same affinity given only the values and weights of the single-dimension bits. Note also that the first three steps of the algorithm (i.e. the encoding) are exactly the same as ordinary spectral hashing (SH) and the only *algorithmic difference* is in the retrieval algorithm, which uses the kernel trick to calculate with an exponentially large number of bits. But MDSH is very different from SH in terms of the *formulation* (equation 2). As discussed in section 2, the optimal code under the MDSH formulation will faithfully reproduce the affinity W as the number of bits approaches infinity. In contrast, SH which seeks bits that are independent and minimize the expected Hamming distance between neighbors, may actually give inferior performance as the number of bits increases.

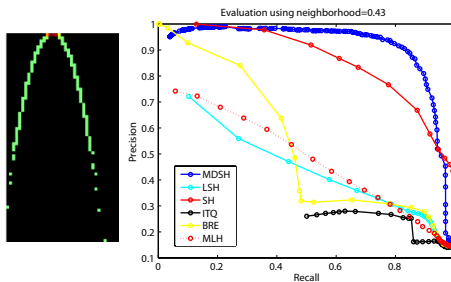


Fig. 4. Left: 2D projection of 32D set constructed to be extremely nonseparable. **Right:** MDSH still outperforms the other methods since it is approximating the correct operator but with an incorrect weighting on the error.

When the input distribution $\Pr(x)$ is separable and the eigenfunctions happen to be binary valued, the MDSH algorithm will give the optimal rank k_2 binary factorization of the affinity matrix, where $k_2 \gg k$ is the number of eigenfunctions with eigenvalue greater than λ_k . For real problems, of course, the input distribution is not guaranteed to be separable nor are the eigenfunctions guaranteed to be binary valued. To deal with the separability assumption, we first apply PCA to the data so that different dimensions are at least uncorrelated (for Gaussian distributions this would also make the distribution separable). Note however, that even if the original distribution is non-separable, the eigenfunctions that assume separability are still the optimal solution to an operator factorization problem (eq. 3) with the correct operator but the wrong density. As long as the separable density gives nonzero weight to points that had nonzero weight in the original density, the eigenfunction expansion will converge to the correct affinity as k increases. In other words, if the eigenfunctions happen to be binary, the MDSH algorithm will converge to the correct affinity even if the distribution is nonseparable.

The binary assumption is more problematic, as can be expected from the fact that binary matrix factorization is intractable. We follow the classic approach of spectral methods (e.g. [19]) and threshold the nonbinary eigenfunctions to obtain an approximate solution to the binary problem. The only exception is when the number of eigenfunctions with significant eigenvalue is less than the number of desired bits. In such cases (which we determine by counting the number of eigenvalues greater than a threshold of 0.1 times the maximal eigenvalue), MDSH cannot create enough bits with significant weight and we concatenate to the current code a second code that uses the exact same eigenfunctions but instead of thresholding at zero, thresholds at another value. We repeat this concatenation until the number of bits is equal to the bit budget.

We use the code from [16] to numerically calculate the eigenfunctions for a single dimension, where we use the fact that generalized eigenfunctions of the graph Laplacian (which are what [16] solves) correspond to eigenfunctions of a reweighted affinity operator.

3 Results

We first revisit the toy data shown in the introduction. This data is sampled from a 32D Gaussian with the standard deviation along the i th dimension equal to $1/i^2$. We compare MDSH to five existing approaches [6, 2, 12, 9, 10]. In all comparisons, we use the code made available by the authors of the paper. Figure 3 shows that MDSH, which explicitly tries to model the affinity with a given σ outperforms the other algorithms for different regimes. Note also that the performance of SH, which ignores the “outer-product” eigenfunctions, may decrease with more bits but MDSH which does not ignore these eigenfunctions continues to improve.

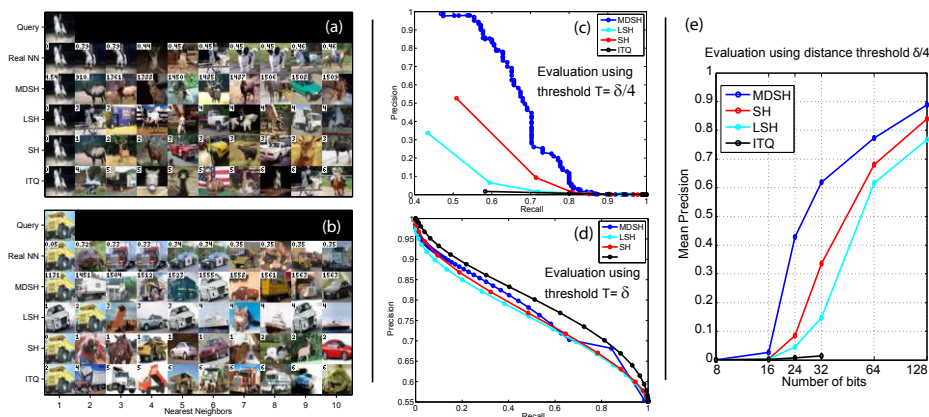


Fig. 5. Results on CIFAR dataset, 64K images. **Left:** 2 randomly selected test images and the retrieved neighbors. **Middle:** Precision-recall curves for different threshold. **Right:** Mean precision as a function of number of bits.

To check the robustness to the separability assumption, we take the 32D synthetic data and set $x(i) = x^2(i - 1)$ for all even dimensions. This creates strong, nonlinear dependencies in the data which PCA cannot undo, thus the separability assumption does not hold. Nevertheless, we find that MDSH still outperforms the other methods (Figure 4). As discussed in section 2.3, even with the separability assumption fails, MDSH is still approximating the correct operator, but with an incorrect weight function on the error.

Our next experiment uses the exact same CIFAR dataset used in [12]. It consist of 64,800 images from 10 ground-truth classes. We use 32 PCA coefficients of the GIST descriptors and show results when the “distance threshold”, i.e. the definition of ground-truth neighbors is set to δ , the average distance between any two images, and when the distance threshold is set to $\delta/4$. As can be seen, for large distance thresholds, all algorithms perform well but the ITQ algorithm works best. In contrast, in the more challenging regime of small distance thresholds, the MDSH algorithm outperforms all other algorithms. BRE

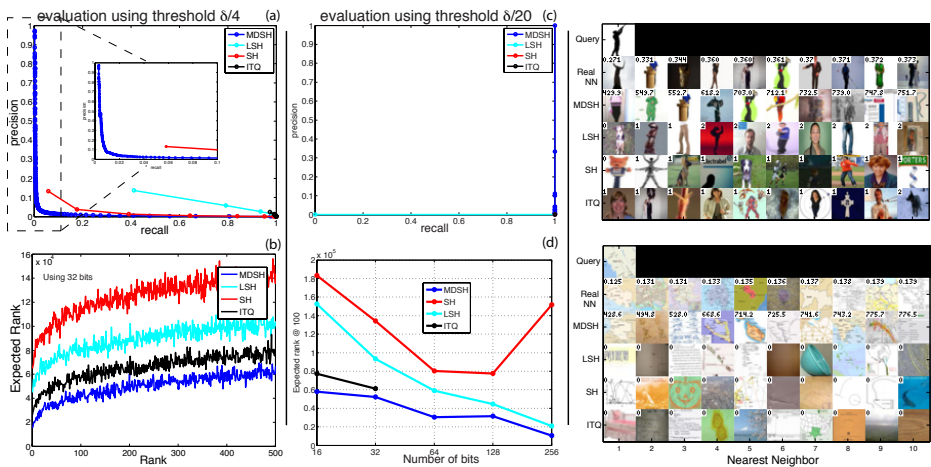


Fig. 6. Results on 1 million Tiny images with 32 bits. **Top and Middle left:** precision recall curve with $T = \delta/4, T = \delta/20$. **Bottom left:** The average rank in terms of Euclidean similarity for the k th neighbor retrieved by each code. Low curves are better. **Bottom middle:** Expected rank at 100 as a function of number of bits. Low curves are better. **Right:** 2 randomly selected test images and the retrieved neighbors.

and MLH were very slow in these runs so were not compared. Note that ITQ cannot generate more bits than dimensions in the input (32 dims).

Finally, we show results on a database of 1 Million Tiny Images (Figure 6). Unlike the CIFAR dataset which is structured into 10 groups by construction, in this unstructured dataset we find that the setting of a reasonable distance threshold is very tricky— since almost all pairs of points are at distance δ . When we set the threshold to be $\delta/20$, MDSH performs perfectly and retrieves all similar pairs with perfect precision. For threshold $\delta/4$, it still has high precision but returns only a small fraction of the retrieved pairs. In contrast LSH and ITQ simply retrieve a huge number of pairs even for the smallest possible threshold and therefore have very low precision in both cases.

To avoid the dependence on setting the distance threshold, we present in the bottom left figure a different measure of performance. For each algorithm and number k we find the k th nearest neighbor defined by the code and find the ranking of that image in the original gist distance. The lower the curve the better. MDSH is the best using this measure (which does not require a distance threshold). The middle bottom figure shows that MDSH continues to outperform the other methods using this measure for different number of bits.

4 Discussion

What makes a good similarity-preserving binary code? One option is to define a code as good if Hamming distance matches the original Euclidean distance. In this paper, we have shown that this definition may be problematic, since in

applications where we care about retrieving “similar” neighbors, matching all distances exactly may be a waste of bits. Instead we have suggested an alternative criterion, where the goal is to have Hamming affinity match the desired affinity matrix. We have shown that this leads to an intractable, binary matrix factorization problem but that the spectral relaxation can be used to build excellent codes with very simple learning algorithms.

Acknowledgements

This work was supported by the ISF and by a Sloan Research Fellowship.

References

1. Gionis, A., Indyk, P., Motwani, R.: Similarity Search in High Dimensions via Hashing. In: Proc. Intl Conf. on Very Large Data Bases. (1999)
2. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: FOCS. (2006) 459–468
3. Raginski, M., Lazebnik, S.: Locality-sensitive binary codes from shift-invariant kernels. In: NIPS. (2009)
4. Salakhutdinov, R.R., Hinton, G.E.: Semantic hashing. In: SIGIR workshop on Information Retrieval and applications of Graphical Models. (2007)
5. Torralba, A., Fergus, R., Weiss, Y.: Small Codes and Large Image Databases for Recognition. In: CVPR. (2008)
6. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS. (2008)
7. Xu, H., Wang, J., Li, Z., Zeng, G., Li, S., Yu, N.: Complementary hashing for approximate nearest neighbor search. In: ICCV. (2011) 1631–1638
8. Wang, J., Kumar, S., Chang, S.F.: Sequential projection learning for hashing with compact codes. In: ICML. (2010) 1127–1134
9. Kulis, B., Darrell, T.: Learning to Hash with Binary Reconstructive Embeddings. In: NIPS. (2009)
10. Norouzi, M., Fleet, D.: Minimal loss hashing for compact binary codes. In: ICML. (2011)
11. R. Lin, D.R., Yagnik, J.: Spec hashing: Similarity preserving algorithm for entropy-based coding. In: CVPR. (2010)
12. Gong, Y., Lazebnik, S.: Iterative quantization: A procrustean approach to learning binary codes. In: CVPR. (2011)
13. Srebro, N., Jaakkola, T.: Weighted low-rank approximations. In: ICML. (2003) 720–727
14. Kumar, S., Mohri, M., Talwalkar, A.: Sampling techniques for the Nystrom method. In: AISTATS. (2009)
15. Coifman, R.R., Lafon, S., Lee, A., Maggioni, M., Nadler, B., Warner, F., Zucker, S.: Geometric diffusion as a tool for harmonic analysis and structure definition of data, part i: Diffusion maps. PNAS **21** (2005) 7426–7431
16. Fergus, R., Weiss, Y., Torralba, A.: Semi-supervised learning in gigantic image collections. In: NIPS. (2009)
17. Belkin, M., Niyogi, P.: Towards a theoretical foundation for laplacian-based manifold methods. J. Comput. Syst. Sci. **74** (2008) 1289–1308
18. Nadler, B., Srebro, N., Zhou, X.: Statistical analysis of semi-supervised learning: The limit of infinite unlabelled data. In: NIPS. (2009) 1330–1338
19. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE PAMI **22** (2000) 888–905