Proximity Preserving Binary Code Using Signed Graph-Cut

Inbal Lavi,[1] **Shai Avidan,**[1] **Yoram Singer,**[2] **Yacov Hel-Or**[3]
[1]Department of Electrical Engineering, Tel-Aviv University, Israel
[2]Department of Computer Science, Princeton University, NJ, USA
[3]School of Computer Science, The Interdisciplinary Center, Israel
inballavi@mail.tau.ac.il, toky@idc.ac.il

Abstract

We introduce a binary embedding framework, called *Proximity Preserving Code (PPC)*, which learns similarity and dissimilarity between data points to create a compact and affinity-preserving binary code. This code can be used to apply fast and memory-efficient approximation to nearest-neighbor searches. Our framework is flexible, enabling different proximity definitions between data points. In contrast to previous methods that extract binary codes based on unsigned graph partitioning, our system models the attractive and repulsive forces in the data by incorporating positive and negative graph weights. The proposed framework is shown to boil down to finding the minimal cut of a signed graph, a problem known to be NP-hard. We offer an efficient approximation and achieve superior results by constructing the code bit after bit. We show that the proposed approximation is superior to the commonly used spectral methods with respect to both accuracy and complexity. Thus, it is useful for many other problems that can be translated into signed graph cut.

Introduction

Content-based image retrieval is a fundamental problem in computer vision, media indexing, and data analysis. A common solution to the problem consists of assigning each image an indicative feature vector and retrieving similar images by defining a distance metric in the feature vector space.

One of the successful uses of deep learning is *data embedding* (Chopra, Hadsell, and LeCun 2005; Koch, Zemel, and Salakhutdinov 2015), where a network is used to map input data into a feature vector space, satisfying some desired distance properties. This technique has many applications, such as word embedding for machine translation (Mikolov et al. 2013), face embedding for identity recognition (Taigman et al. 2014; Schroff, Kalenichenko, and Philbin 2015; Wen et al. 2016; Liu et al. 2017), and many more. The main idea behind data embedding is to find a mapping from input space into a vector space where the distances in the embedding space conform with the desired task.

In a typical scenario, the embedding space is several hundreds of bytes long (e.g., 512 bytes in FaceNet (Schroff, Kalenichenko, and Philbin 2015) embedding), and a new query may be compared to the existing images by nearest-neighbor (NN) search. As the number of images scales up, the memory required to store all the examples becomes too large, and the time complexity to apply NN search becomes a critical bottleneck.

Many solutions have been proposed to mitigate this issue, including dimensionality reduction (Liu and Schisterman 2004) and approximate NN search (Muja and Lowe 2014). In recent years, a family of algorithms called *Binary Hashing* or *Hamming Embedding* has gained popularity. These algorithms find a mapping from a feature space into a Hamming space using a variety of methods. The main advantages of a binary representation are the significant reduction in storage and in the time required to apply vector comparisons: vectors are compared not in high-dimensional Euclidean space, but rather in the Hamming space, utilizing the extremely fast XOR operation. This representation is highly valuable in mobile systems, as on-device training is limited due to computational shortage. This requires ad-hoc hashing methods that can be computed on simple hardware, and that can be generalized well to novel data points.

Many modern Hamming embedding techniques are data-dependent. Data-dependent methods work by learning an affinity matrix between data points while attempting to preserve their affinities in Hamming space. *In-sampled* techniques aim at generating a set of binary codes, a single code for each data point, whose Hamming distances conform with the affinity matrix. *Out-of-sample* techniques deal with novel samples that are not known in advance. These techniques learn a general functional mapping that maps query points from feature space into Hamming space.

Affinity between data pairs can be, for example, related to the metric distances between their associated features, or semantic relations indicating data points belonging to the same semantic class. The affinity matrix is usually relaxed to *positive* values, where small values indicate weak proximity (far pairs), and large values indicate strong proximity (near pairs). This encourages near pairs to be located close by in the Hamming space but does not constrain the far pairs.

We propose a binary hashing framework called *Proximity Preserving Code* (PPC). The main contribution of our method

is that the binary code is constructed based on positive and negative proximity values, representing attractive and repulsive forces. These forces properly arrange the points in the Hamming space while respecting the pairwise affinities. Our solution models this proximity as a signed graph, and the code is computed by finding the min-cut of the graph. This problem can be formulated as the *max-cut* problem (due to the negative values) and is known to be NP-hard (Alon and Naor 2004). We demonstrate that our approach is more accurate and memory-efficient as compared to state of the art graph-based embeddings.

Previous Works

Previous works in Hamming embedding can be classified into two distinct categories: data-independent and data-dependent. Data-independent methods are composed of various techniques for dimensionality reduction or techniques for dividing the N-dimensional space into buckets with equal distributions. One of the most popular data-independent hashing methods is Locality Sensitive Hashing (LSH) (Datar et al. 2004). LSH is a family of hash functions that map similar items to the same hash bucket with higher probability than dissimilar items.

Data-dependent methods learn the distribution of the data in order to create accurate and efficient mapping functions. These functions are usually comprised of three elements: the hash function, the similarity measure, and an optimization criterion. Hash functions vary and include linear functions (Norouzi and Blei 2011), nearest vector assignment (He, Wen, and Sun 2013), kernel functions (Kulis and Darrell 2009), neural networks (Lin et al. 2015), and more. Similarity measures include Hamming distance and different variants of Euclidean or other compute-intensive distances that are precomputed for vector assignment (Jegou, Douze, and Schmid 2011). Optimization criteria mainly use variants of similarity preservation and code balancing. We will focus on binary hashing methods.

An influential work in binary hashing methods is Spectral Hashing (Weiss, Torralba, and Fergus 2009). This method creates code words $\{\mathbf{c}_i\}$ that preserve the data similarity. By defining an affinity matrix $W_{ij} = exp(-\frac{\|x_i - x_j\|^2}{\epsilon^2})$, the authors turn the hashing problems into a minimization of $\sum_{ij} W_{ij}\|\mathbf{c}_i - \mathbf{c}_j\|^2$, subject to: $\mathbf{c}_i \in \{-1, 1\}^k$, $\sum_i \mathbf{c}_i = \mathbf{0}$ (code balancing), and $\frac{1}{n}\sum_i \mathbf{c}_i\mathbf{c}_i^T = I$ (independence). This minimization problem for a single bit can be cast as a graph partitioning problem which is known to be NP-hard. A good approximation for this problem is achieved by using spectral methods. The code is obtained by computing the k eigenvectors corresponding to the smallest eigenvalues of the graph Laplacian of W and thresholding them at zero.

Liu et al. (2011) proposed the Anchor Graph Hashing, a hashing method utilizing the idea of a low-rank matrix that approximates the affinity matrix, to allow a graph Laplacian solution that is scalable both for training and out-of-sample computation. Shen et al. (2013) present Inductive Manifold Hashing, a method that learns a manifold from the data and utilizes it to find a Hamming embedding. They demonstrate their results with several approaches, including Laplacian eigen-maps and t-SNE. Shen et al. (2015) and Liu et al. (2014) directly optimize the discrete problem, and employ discrete coordinate descent to achieve better precision on the graph problem. Scalable Graph Hashing with Feature Transformation (Jiang and Li 2015) uses a feature transformation method to approximate the affinity graph, allowing faster computation on large scale datasets. They also proposed a sequential approach to learn the code bit-by-bit, allowing for error-correcting of the previously computed bits. Li, Hu, and Nie (2017) revisit the spectral solution to the Laplacian graph and propose a spectral rotation that improves the accuracy of the solutions.

All of the above approaches formulate the graph Laplacian by defining an affinity matrix that takes into account the similarities between points in the training set. However, they do not address the dis-similarity, or push-pull forces in the data set. In this paper, we propose a binary embedding method that employs an affinity matrix of both positive and negative values. We argue that this type of affinity better represents the relationships between data points, allowing a more accurate code generation. The characteristics and the advantages of this work are as follows:

- Our code is constructed by solving a signed graph-cut problem, to which we propose a highly accurate solution. We demonstrate that the signed graph provides a better encoding for the forces existing in the coding optimization. We show that the commonly used spectral solution, which works well in the unsigned graph-cut problems, is unnecessary, costly, and inferior in this scenario.
- The code is computed one bit at a time, allowing for error correction during the construction of the hashing functions.
- We split the optimization into two steps. We first optimize for a binary vector representing the in-sample data, and then we fit the hashing functions to obtain accurate code for out-of-sample points.
- Our framework is flexible, allowing various proximity definitions, including semantic proximity. This can be useful for many applications, especially in low computation environments.

Problem Formulation

We are given a set of n data points $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$, $\mathbf{x}_i \in \mathbb{R}^d$, in some vector space, and a proximity relation between pairs of points $(i, j) \in \mathcal{S}$, where $\mathcal{S} = \{1..n\} \times \{1..n\}$. We assign each pair of points in $\mathcal{S}$ to be in the *Near* or *Far* group, according to some proximity measure. This proximity measure can have a semantic meaning, geometric meaning, or any other adjacency relation. Formally, we define

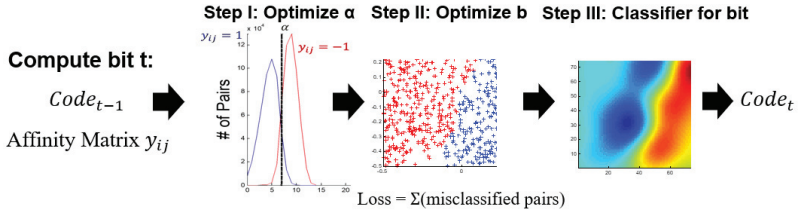
$$\mathcal{N} = \{(i, j) \mid \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are in the same class}\}$$

and

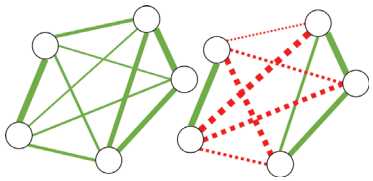
$$\mathcal{F} = \{(i, j) \mid \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are in different classes}\}$$

Note that $\mathcal{N}$ and $\mathcal{F}$ induce a partition of $\mathcal{S}$ into two disjoint sets: $\mathcal{N} \cup \mathcal{F} = \mathcal{S}$ where $\mathcal{N} \cap \mathcal{F} = \emptyset$.

In a classification scenario, for example, two points belonging to the same class will be defined as *Near*; otherwise,



(a) PPC Algorithm



(b) Unsigned vs. signed graph

Figure 1: (a) PPC algorithm overview. To compute the t^{th} bit, we first find the optimal α for the existing code ($t-1$ bits), then compute a new bit $\mathbf{b}$ by minimizing the loss $\mathcal{E}$. For this bit, we compute a binary classifier $h^t(\mathbf{x})$ that is used as the t^{th} hashing function. (b) Illustration of an unsigned graph (left) vs. a signed graph (right) describing the same relations between nodes. The graph edges in green (solid line) are edges with positive weights, and the red (dashed lines) are edges with negative weights. The line thickness indicates the weight magnitude.

they will be defined as *Far*. Another example of an adjacency matrix is a neighborhood of a certain radius. For a distance metric $d_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)$ in $\mathbb{R}^d$ and a given radius r we define:

$$\mathcal{N} = \{(i,j) \mid d_{ij} \leq r\} \quad \text{and} \quad \mathcal{F} = \{(i,j) \mid d_{ij} > r\} \quad (1)$$

Denote the p-length binary code of point $\mathbf{x}_i$ by $\mathbf{c}_i \in \{\pm 1\}^p$. Our goal is to find n binary codes $\{\mathbf{c}_i\}_{i=1}^n$ that satisfy the following two requirements:

- *Compactness*: The length of the code should be short, i.e., p should be as small as possible.
- *Proximity Preserving*: The binary code should preserve the proximity of $\mathcal{X}$. That is, there exists a constant α s.t. $d_H(\mathbf{c}_i, \mathbf{c}_j) \leq \alpha$ for each pair $(i,j) \in \mathcal{N}$, and $d_H(\mathbf{c}_i, \mathbf{c}_j) > \alpha$ for each $(i,j) \in \mathcal{F}$, where $d_H(\cdot,\cdot)$ stands for the Hamming distance between two binary codes[1]:

$$d_H(\mathbf{c}_i, \mathbf{c}_j) = \sum_{k=1}^{p} (1 - \mathbf{c}_i[k]\mathbf{c}_j[k]) = (p - \mathbf{c}_i^T \mathbf{c}_j).$$

It can be shown that if proximity relationships are determined according to ℓ_1 or ℓ_2 distance between points in $\mathbb{R}^d$, the *Proximity Preserving* requirement can be fully satisfied using large enough codes (i.e., p is large). However, due to the compactness requirement, we want to relax the proximity preserving requirement and try to find an optimal code for a given code length.

Denote a *proximity label*, y_{ij}, associated with each pair of points $(i,j) \in \mathcal{S}$:

$$y_{ij} = \begin{cases} +1 & \text{if } (i,j) \in \mathcal{N} \\ -1 & \text{if } (i,j) \in \mathcal{F} \end{cases}$$

For a given value $\alpha > 0$ we define:

$$z_{ij} = y_{ij}(\alpha - d_H(\mathbf{c}_i, \mathbf{c}_j)) \,. \quad (2)$$

We would like that for each pair (i,j), $z_{ij} \geq 0$, and accordingly we define a loss function:

$$l(z_{ij}) = \begin{cases} 1 & \text{if } z_{ij} < 0 \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

[1]In fact, this definition is twice the Hamming distance, but we stick with it for sake of clarity.

The empirical loss for the entire set reads:

$$\mathcal{E}(\{y_{ij}\}, \{\mathbf{c}_i\}) = \min_{\alpha} \sum_{(i,j)\in\mathcal{S}} l(z_{ij}) \quad (4)$$

This loss penalizes pairs of points that are mislabeled, that is, pairs of points in $\mathcal{F}$ whose Hamming distance is smaller than α, or pairs of points in $\mathcal{N}$ whose Hamming distance is larger than α.

Definition 1 (Proximity Preserving Code)**.** *Given a set of data points $\mathcal{X}$ along with their proximity labels, $\{y_{ij}\}$, a* Proximity Preserving Code *(PPC) of length p is a binary code, $\{\mathbf{c}_i\}_{i=1}^n$, $\mathbf{c}_i \in \{\pm 1\}^p$, that minimizes $\mathcal{E}(\{y_{ij}\}, \{\mathbf{c}_i\})$.*

In the following we describe the procedure to generate the PPC. In particular, we show that finding PPC for a given set of points boils down to applying an integer low-rank matrix decomposition. We provide two possible approximated solutions and show their connection to the minimum signed graph-cut problem. Finally, we provide a solution for extracting hashing functions for out-of-sample data points.

Proximity Preserving Code

Recall the definition of z_{ij} (Equation 2): $z_{ij} = y_{ij}(\alpha - d_H(\mathbf{c}_i, \mathbf{c}_j))$. Substituting the Hamming distance into this expression we get:

$$z_{ij} = y_{ij}\left(\alpha - \left(p - \mathbf{c}_i^T \mathbf{c}_j\right)\right) = y_{ij}\left(\mathbf{c}_i^T \mathbf{c}_j - \beta\right) \quad (5)$$

where we define $\beta = p - \alpha$.

To simplify notations we define a *code matrix* $C \in \{\pm 1\}^{p \times n}$ by stacking the code words along its columns:

$$C = \begin{pmatrix} | & | & & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ | & | & & | \end{pmatrix}$$

Similarly we define

$$B = C^T C \quad \text{where} \quad B_{ij} = \mathbf{c}_i^T \mathbf{c}_j$$

Equation 5 can now be defined over the entries of matrix B:

$$z_{ij} = y_{ij}(B_{ij} - \beta)$$

and the total loss (Equation 4) is:

$$\mathcal{E}(\{y_{ij}\}, B) = \min_{\beta} \sum_{ij} l(z_{ij}) \tag{6}$$

Denote by $\mathbf{b}^k$ the **rows** of C (similarly, the columns of C^T) such that

$$C^T = \begin{pmatrix} | & | & & | \\ \mathbf{b}^1 & \mathbf{b}^2 & \cdots & \mathbf{b}^p \\ | & | & & | \end{pmatrix}$$

Each $\mathbf{b}^k \in \{\pm 1\}^n$ is a vector representing the k^{th} bit of all the code words (n words). The matrix $B = C^T C$ can now be represented as a linear sum of $n \times n$ matrices:

$$B = \sum_{k=1}^{p} \mathbf{b}^k \mathbf{b}^{k^T} = \sum_{k=1}^{p} B^k \tag{7}$$

where $B^k = \mathbf{b}^k \mathbf{b}^{k^T}$ is a rank-1 matrix extracted from the k^{th} bit of the code words. Thus, each additional bit can either increase the rank of matrix B or leave it the same. Our goal then is to find a low rank matrix $B = C^T C$, minimizing the loss defined in Equation 6.

The minimization function defined in Equation 6 introduces a combinatorial problem which is NP-hard. Therefore we relax the binary loss function and re-define it using a logistic loss function:

$$l(z_{ij}) = \tilde{\ell}(y_{ij}(B_{ij} - \beta))$$

where $\tilde{\ell}(z) = \ln(1 + e^{-z})$. The relaxed total loss is therefore

$$\mathcal{E}(\{y_{ij}\}, B) = \min_{\beta} \sum_{ij} \tilde{\ell}(y_{ij}(B_{ij} - \beta)) \tag{8}$$

Bit Optimization

In the proposed process we generate the codes for n data points in a sequential manner, bit after bit. In the following we detail the minimization process for bit k. This is also illustrated in Figure 1a.

At this step we assume that $k-1$ bits of PPC code have already been generated. Denote

$$B^{1:k} = \sum_{\ell=1}^{k} B^{\ell} \;\text{ where }\; B^{\ell} = \mathbf{b}^{\ell} \mathbf{b}^{\ell^T}.$$

For the k^{th} bit, we minimize Equation 8 with respect to B^k and β as follows:

$$\mathcal{E}^k = \sum_{ij} \tilde{\ell}(y_{ij}(B_{ij}^{1:k-1} + B_{ij}^k - \beta)) \tag{9}$$

Note that $B_{ij}^{1:k-1}$ is already known at step k. As mentioned above, $\mathcal{E}^k$ is minimized using alternate minimization, described below.

Step I - optimizing β:

$\mathcal{E}^k$ is convex with respect to β, so any scalar search is applicable here. Since the loss $\tilde{\ell}(z)$ is nearly linear for $z \leq 0$, a fast yet sufficiently accurate approximation for β is to choose the value that equates the number of misclassified pairs in the $\mathcal{N}$ and $\mathcal{F}$ sets. For the current code $\{\mathbf{c}_i\}_{i=1}^n$, $\mathbf{c}_i \in \{\pm 1\}^{k-1}$, and a constant value α, define the misclassified sets:

$$E_N(\alpha) = \{(i,j) \mid (i,j) \in \mathcal{N} \text{ and } d_H(\mathbf{c}_i, \mathbf{c}_j) > \alpha\}$$

and similarly

$$E_F(\alpha) = \{(i,j) \mid (i,j) \in F \text{ and } d_H(\mathbf{c}_i, \mathbf{c}_j) \leq \alpha\}$$

The value of α is set such that the cardinality of the two sets is equal, i.e., the $\hat{\alpha}$ that satisfies:

$$|E_N(\hat{\alpha})| = |E_F(\hat{\alpha})| \tag{10}$$

and accordingly $\hat{\beta} = (t-1) - \hat{\alpha}$. This is visualized in Step I of Figure 1a. We show a histogram of the near pairs of samples in blue and the far pairs in red, and α is the vertical black line thresholding the Hamming distance.

Step II - optimizing $\mathbf{b}^k$:

For the evaluated $\hat{\beta}$, Equation 9 becomes:

$$\begin{aligned} \mathcal{E}^k &= \sum_{ij} \tilde{\ell}(y_{ij}(B_{ij}^{1:k-1} + B_{ij}^k - \hat{\beta})) \\ &= \sum_{ij} \tilde{\ell}(y_{ij}(B_{ij}^k + \gamma_{ij}^{k-1})) \end{aligned}$$

where we define $\gamma_{ij}^{k-1} = B_{ij}^{1:k-1} - \hat{\beta}$. In a forward greedy selection process, we approximate the potential decrease in the loss using the gradient. Our goal is to find B^k that minimizes $\mathcal{E}^k \approx \mathcal{E}^{k-1} + \Delta\mathcal{E}^k$ or alternatively maximizes $-\Delta\mathcal{E}^k$ where:

$$-\Delta\mathcal{E}^k = -\sum_{ij} \frac{\partial \mathcal{E}^k}{\partial B_{ij}^k} B_{ij}^k = -\sum_{ij} \tilde{\ell}'(y_{ij}\gamma_{ij}^{k-1}) y_{ij} B_{ij}^k$$

where $\tilde{\ell}'$ stands for the derivative of the logistic loss function $\tilde{\ell}'(z) = -1/(1+e^z)$.

Defining $w_{ij} = -y_{ij}\tilde{\ell}'(y_{ij}\gamma_{ij}^{k-1})$, we arrive at the following maximization problem:

$$\max_{B^k} \sum_{ij} w_{ij} B_{ij}^k = \max_{\mathbf{b}^k} \sum_{ij} w_{ij} \mathbf{b}^k[i] \mathbf{b}^k[j]$$

where the maximization is taken over all entries of $\mathbf{b}^k \in \{\pm 1\}^n$. For the sake of simplicity we omit the superscript k and denote $\mathbf{b}^k$ by $\mathbf{b}$. Collecting $\{w_{ij}\}$ into matrix W, s.t. $W(i,j) = w_{ij}$, the above maximization can be simply expressed in a matrix form:

$$\hat{\mathbf{b}} = \underset{\mathbf{b}}{\operatorname{argmax}}\, \mathbf{b}^T W \mathbf{b} \quad \text{s.t. } \mathbf{b} \in \{\pm 1\}^n \tag{11}$$

If the weight matrix W was all positive (all entries are positive values), this problem can be interpreted as a graph min-cut problem. In our problem, however, the matrix W is comprised of both positive and negative values, indicating pairs (i,j) that are properly and improperly assigned as near or far according to the code computed. This is termed in the literature a *signed min-cut* problem which is equivalent to the *max-cut* problem whose solution is NP-hard.

In the proposed solution we start with an initial guess for the bit vector $\mathbf{b}$ and improve it by using a forward greedy selection scheme. We present two iterative approaches for the selection scheme: *vector update* and *bit update*.

Vector Update Given an initial guess for $\mathbf{b}$, the *vector update* method updates the entire vector at once. At each iteration the vector is improved by applying:

$$\mathbf{b}' = sign(W\mathbf{b})$$

where $\mathbf{b}'$ is the updated vector that satisfies: $\mathbf{b}'^T W\mathbf{b}' \geq \mathbf{b}^T W\mathbf{b}$. The following four theorems prove that $\mathbf{b}'$ is a better vector than $\mathbf{b}$:

Theorem 1. *For any $n \times n$ matrix W and $\mathbf{b}, \mathbf{b}' \in \{\pm 1\}^n$, if $\mathbf{b}' = sign(W\mathbf{b})$, then $\mathbf{b}'^T W\mathbf{b} \geq \mathbf{b}^T W\mathbf{b}$*

Theorem 2. *Assuming W is Positive Semidefinite, if $\mathbf{b}'^T W\mathbf{b} > \mathbf{b}^T W\mathbf{b}$, then $\mathbf{b}'^T W\mathbf{b}' > \mathbf{b}^T W\mathbf{b}$.*

Theorem 3. *Any symmetric matrix W can become Positive Semidefinite by applying $W \leftarrow W + |\lambda|I$ where λ is the smallest eigenvalue of W.*

Theorem 4. *Adding a constant value to the diagonal of the weight matrix W will not affect the output code computed.*

Proofs are given in the Appendix. Using the above theorems, Algorithm 1 summarizes the *vector update* iterations.

Algorithm 1 Vector Update $(\mathbf{b}, W)$

$\hat{\mathbf{b}} \leftarrow \mathbf{b}$
$W \leftarrow W + |\lambda|I$ where λ is the smallest e.v. of W
repeat
  $\mathbf{b} \leftarrow \hat{\mathbf{b}}$
  $\hat{\mathbf{b}} \leftarrow sign(W\mathbf{b})$
until $\hat{\mathbf{b}} = \mathbf{b}$
return $\mathbf{b}$

Bit Update Unlike the *vector update*, the *bit update* method changes one bit at a time: For each bit in vector $\mathbf{b}$, we flip the bit and determine whether the new value improved the objective $\mathbf{b}^T W\mathbf{b}$. This is repeated for each bit sequentially, and over the entire vector, until convergence. This procedure can be applied very efficiently using the following scheme: Define $\mathbf{b} = \mathbf{b}_{(i)} + \mathbf{b}_{(-i)}$, where $\mathbf{b}_{(i)} = (0...b_i...0)$ is a one-hot vector with the i^{th} entry of $\mathbf{b}$ at the i^{th} coordinate. Accordingly, $\mathbf{b}_{(-i)} = \mathbf{b} - \mathbf{b}_{(i)}$ is the vector $\mathbf{b}$ with 0 at the i^{th} coordinate. When optimizing the i^{th} bit:

$$\begin{aligned}\mathbf{b}^T W\mathbf{b} &= (\mathbf{b}_{(i)} + \mathbf{b}_{(-i)})^T W(\mathbf{b}_{(i)} + \mathbf{b}_{(-i)}) = \\ &= \mathbf{b}_{(i)}^2 W(i,i) + 2\mathbf{b}_{(-i)}^T W\mathbf{b}_{(i)} + const\end{aligned}$$

It can be verified that the only term affecting the optimization is $\mathbf{b}_{(-i)}^T W\mathbf{b}_{(i)}$. Therefore we can optimize each bit in $\mathbf{b}$ by looking at the value of the i^{th} element of $\mathbf{b}_{(-i)}^T W$. Subsequently, the only elements affecting this value in the matrix W are in the i^{th} column of W. Thus,

$$\mathbf{b}[i]' = sign\left(\mathbf{b}_{(-i)}^T W[:,i]\right) \tag{12}$$

where we denote by $W[:,i]$ the i^{th} column of W. We apply this optimization scheme for each bit sequentially, and repeatedly over the entire vector $\mathbf{b}$, until convergence. Each bit update is inserted immediately into $\mathbf{b}$ so that the optimization for the $i+1$ bit will account for the preceding bits that have been calculated. This update is computationally inexpensive, requiring $O(n)$ operations for each bit update, and $O(n^2)$ operations for one round over the entire $\mathbf{b}$. This method is summarized in Algorithm 2.

Algorithm 2 Bit Update $(\mathbf{b}, W)$

repeat
  $\hat{\mathbf{b}} \leftarrow \mathbf{b}$
  for $i \leftarrow 1, N$ **do**
    $\mathbf{b}_{-i} \leftarrow \mathbf{b}, \quad \mathbf{b}_{-i}[i] \leftarrow 0$
    $\mathbf{b}[i] \leftarrow sign(\mathbf{b}_{-i}^T W[:,i])$
until $\hat{\mathbf{b}} = \mathbf{b}$
return $\mathbf{b}$

The two algorithms presented for the iterative bit optimization scheme provide a solution to the max-cut problem where both positive and negative weights appear on the graph edges. The iterations require several light computations and stop when a local maximum is reached and the iteration scheme can no longer improve upon the current bit vector. We show in the Experiments Section that the bit update scheme achieves better codes than the vector update and is therefore preferable.

Initial Guess

Our method is based on an iterative scheme. Therefore, we start the optimization with an initial guess and improve upon it. A common solution is to relax the constraints $\mathbf{b}[i] \in \pm 1$ and allow real-valued solutions. This enables the maximization problem to be cast as an eigenvalue problem. The final solution is then obtained by thresholding the results, in a similar manner to Weiss, Torralba, and Fergus (2009). Interestingly, we have found that starting from a *random guess* and applying the suggested iterations produces more accurate solutions and with much faster compute time than the traditional eigenvalue solutions.

In conclusion, the algorithm provided above can be used to solve the signed graph min-cut problem where W consists of both positive and negative weights. We show empirically that our solution is equivalent to or outperforms other methods, by starting from a random guess solution and applying an update scheme until convergence. Note, that the proposed update schemes can improve upon any approximated solution suggested in the literature, as the suggested iterations do not deteriorate and can only improve the objective function. Our evaluations and experimental results are provided in the Experiments Section.

Signed Graph Min-Cut Problem

Equation 11 suggests that our problem can be cast as a signed graph min-cut problem. A weighted graph is represented by a vertex set $V = \{1, \cdots, n\}$ and weights $W_{ij} = W[i,j] = W[j,i]$ for each pair of its vertices $(i,j) \in V \times V$. The weight of the minimum cut $w(G, \bar{G})$ is given by the following

problem:

$$Minimize \quad \frac{1}{2}\sum_{i,j} W_{ij}(1 - b_i b_j) \;\; s.t. \;\; b_i \in \{\pm 1\}. \quad (13)$$

where $\mathbf{b} = [b_1, \cdots, b_n]$ is an indicator vector s.t. $b_i = 1$ if $i \in G$ and $b_i = -1$ if $i \in \bar{G}$. The above minimization is an integer quadratic program whose solution is known to be NP-hard (Alon and Naor 2004). Note that the above formulation can be expressed similarly by $maximize \;\; \mathbf{b}^t W \mathbf{b} \;\; s.t. \;\; \mathbf{b} \in \{\pm 1\}^n$, which is similar to the expression given in Equation 11. The weights collected in Equation 13 refer only to pairs $(i, j) \; s.t. \;\; b_i \neq b_j$. Thus, the minimal cut aims at including as many negative weights as possible while excluding positive weights. Since we are dealing with signed graphs, balancing the cut is not critical as it is in unsigned graphs since cutting a small component with few edges does not necessarily provide the smallest cut.

Alon and Naor (2004) define the above problem as a $\|W\|_{\infty \to 1}$ norm and provide a semidefinite relaxation: $maximize \sum_{ij} W_{ij} \mathbf{u}_i \cdot \mathbf{v}_j$ s.t. $\|\mathbf{u}_i\| = \|\mathbf{v}_j\| = 1$. The semidefinite program can be solved within an additive error of ϵ in polynomial time. Alon and Naor suggested three techniques to round the semidefinite solution into a binary solution ($b_i \in \{\pm 1\}$), which provides an approximation to the original solution up to a constant factor (K_G, called *Grothendieck's constant*, $1.570 \leq K_G \leq 1.782$). In the Experiments Section we show that our iterative update approach can improve over Alon and Naor's solution when taking their solution as an initial guess. Moreover, taking a random guess as an initial solution provides a final solution that is comparable or better, so the benefit of using a costly approximated solution as initial guess is questionable.

The minimization in 13 can be equivalently rewritten in a quadratic form:

$$Minimize \quad \frac{1}{2}\sum_{i,j} W_{ij}(b_i - b_j)^2 \;\; s.t. \;\; b_i \in \{\pm 1\} \quad (14)$$

The matrix form of the above minimization reads: $mimimize \;\; \mathbf{b}^t L \mathbf{b} \;\; s.t. \;\; \mathbf{b} \in \{\pm 1\}^n$, where $L = D - W$ is the Laplacian of W and D is a diagonal matrix $D_{ii} = \sum_j W_{ij}$. The Laplacian of a graph is frequently used for graph clustering or graph-cut using spectral methods. It was shown that taking the second-smallest eigenvector (the Fiedler vector) and thresholding it at zero provides a relaxed approximation for the minimization in Eq. 14 (see Von Luxburg for more details). However, spectral methods commonly deal with positive weights where the matrix W is guaranteed to be positive semidefinite. This is not the situation in our case where eigenvalues might be negative as well.

Kunegis, Lommatzsch, and Bauckhage (2009) suggested an alternative for graph Laplacian for signed-graphs: $\bar{L} = \bar{D} - W$, where $\bar{D}_{ii} = \sum_j \mid W_{ij} \mid$ and proved that $\bar{L}$ is positive semidefinite. However, Knyazev (2017) argue that the signed Laplacian does not give better clustering results than the original definition of Laplacian, even if the graph is signed. We show in our experiments that neither solution works as well as the greedy update scheme suggested in this paper.

Optimizing the Hashing Functions

Finally, we arrive at the out-of-sample extension and explain how to learn hashing functions to encode out-of-sample data points. We found that it is preferable to first optimize for the binary vector $\mathbf{b}^k$, then learn a hashing function $h^k(\mathbf{x})$ requiring $h^k(\mathbf{x}_i) = \mathbf{b}^k[i]$, for $i = 1..n$. Optimizing directly for the hashing function yields a non-linear optimization that often provides inaccurate results. Splitting the optimization into two steps allows each step to be exploited in the best manner. We assume that novel data points will be drawn from the same distribution of the given data $\mathcal{X}$. Therefore, the hashing functions can be optimized using the empirical loss over $\mathcal{X}$.

We denote by $\tilde{\mathbf{b}}^k$ the optimal $\mathbf{b}^k$ resulting from the first step. This vector encodes the optimal binary values for the k^{th} bit (over all data points). We then train a binary classifier $h^k(\mathbf{x}; \Theta)$ over the input pairs $\{(\mathbf{x}_i, \tilde{\mathbf{b}}^k[i])\}$, by minimizing a loss function:

$$\min_{\Theta} \sum_{i=1}^{n} \mathcal{L}(h^k(\mathbf{x}_i, \Theta), \tilde{\mathbf{b}}^k[i])$$

where Θ denotes the classifier's parameters. We use kernel SVM (Scholkopf and Smola 2001) with Gaussian kernels to classify the points $\{\mathbf{x}_i\}$ into ± 1, but any standard classifier can be applied similarly. At step k, we train the hash functions h^k and construct the k^{th} bit for the binary codes: $\mathbf{b}^k = [h^k(\mathbf{x}_1), h^k(\mathbf{x}_2), \cdots, h^k(\mathbf{x}_n)]^T$. This bit is updated immediately in the codes $\{\mathbf{c}_i\}_{i=1}^n$, allowing the $k+1$ bit to account for the errors in $\mathbf{b}^k$. This error correcting scheme is another benefit of the two-step solution.

As we proceed, the algorithm adds more bits to the PPC code. Each additional bit is aimed at decreasing the total loss. The process terminates when the total loss is below a given threshold, or when the number of bits exceeds p.

Experiments and Results

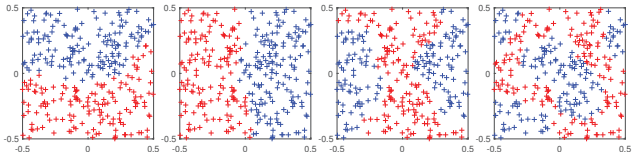


Figure 2: Visualizations of the bits assigned to each data point in a 2-dimensional space according to PPC. In these images, blue points have been assigned 1 while red points have been assigned -1.

To illustrate the optimization process, a synthetic example is shown in Figure 2. In this figure 300 points are drawn in 2D in a range of $[-0.5..0.5] \times [-0.5..0.5]$. The figure shows the first 4 first bits (red/blue indicate ± 1) where the proximity matrix was generated with a r-neighborhood proximity measure. This demonstrates how the PPC algorithm tries to separate the vector space into two labels by balancing between areas with high neighbor density and correcting for the errors of previous bits.

AUC	CIFAR-10							
Code Length	**12**	**16**	**24**	**32**	**48**	**64**	**96**	**128**
SH	0.121379	0.121498	0.119628	0.121108	0.126965	0.12771	0.130133	0.129777
IMH-tSNE	0.154169	0.165783	0.168781	0.150094	0.165669	0.16049	0.163685	0.174634
IMH-LE	0.170701	0.164687	0.144931	0.154949	0.165396	0.152761	0.162143	0.152876
SGH	0.129056	0.12776	0.131998	0.138006	0.136264	0.144698	0.153421	0.157661
LGHSR	0.136739	0.144933	0.149855	0.148962	0.144178	0.144102	0.150296	0.147022
SDH	0.249316	0.191575	0.229962	0.250167	0.227537	0.257303	0.288238	0.326233
PPC	**0.28365**	**0.312302**	**0.308905**	**0.329332**	**0.343186**	**0.352296**	**0.354291**	**0.355555**

Table 1: Area under the curve of precision-recall for varying code lengths on the CIFAR-10 dataset. The methods compared: Spectral Hashing (SH) (Weiss, Torralba, and Fergus 2009), Inductive Manifold Hashing (IMH) (Shen et al. 2013), Scalable Graph Hashing (SGH) (Jiang and Li 2015), Large Graph Hashing with Spectral Rotation (LGHSR) (Li, Hu, and Nie 2017), Supervised Discrete Hashing (SDH) (Shen et al. 2015), and our method (PPC).

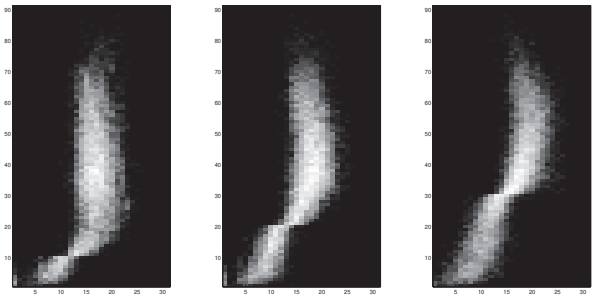


Figure 3: Joint histogram of Hamming distances (x-axis) with respect to the Euclidean distances (y-axis) for PPC code computed on data with r-neighborhood affinity of (left-to-right) $r = 10$, $r = 20$ and $r = 30$.

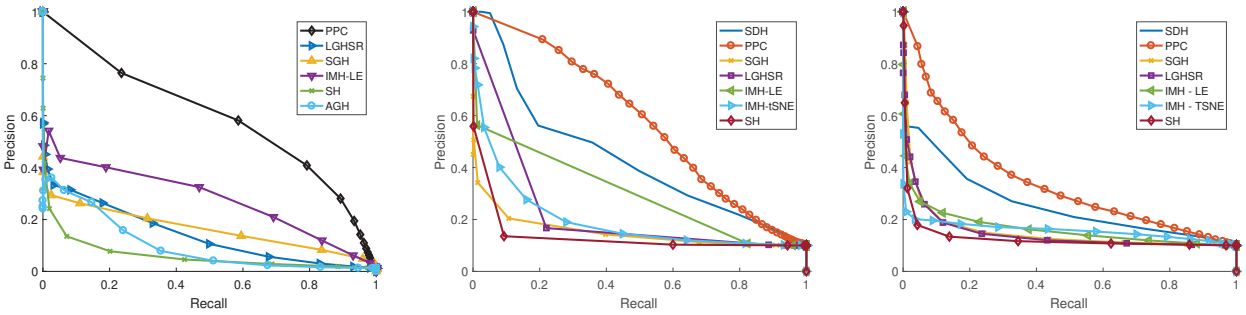
The mutual relationships between the actual distances and the Hamming distances are illustrated in Figure 3, which shows the joint histogram of $d_H(\mathbf{c}_i, \mathbf{c}_j)$ vs. the Euclidean distances, d_{ij} for three cases. This is another synthetic 2D example with varying r-neighborhood proximity measures. The x-axis indicates the Hamming distances of the generated code and the y-axis indicates the actual Euclidean distances. The histograms are plotted as gray-scale images where the gray-value in each entry indicates the number of pairs with the associated distances. The brighter the gray-value, the greater the number of pairs (we display the log of the actual values for a better visualization). For each case we see that most of the pairs with Euclidean distance below $r = 10, 20, 30$ (the pairs labeled as "Near" in this example) are concentrated to the left of the respective Hamming distances, $d_H = 11, 12, 13$. These were also the final α values at the last step of each case. It is interesting to note that the conditional distributions of d_H at the two sides of the alpha values are wide, while the order of the Euclidean distances is not necessarily preserved in the respective Hamming distances. This indicates that the bits are allocated solely to optimize the neighborhood constraints, and not to meet any other requirements such as preserving the ordinal distances. This allows for optimal allocation of the bit resources.

We evaluate Proximity Preserving Code on several public datasets: MNIST (Deng 2012), CIFAR-10 (Krizhevsky, Nair, and Hinton 2014), and LabelMe (Russell et al. 2008). CIFAR-10 (Krizhevsky, Nair, and Hinton 2014) is a labeled subset of the 80 million tiny images dataset, consisting of 60,000 32x32 color images represented by 512-dimensional GIST feature vectors (Oliva and Torralba 2001). It is split into 59,000 images in the training set and 1000 in the test set. MNIST (Deng 2012) is the well-known database of handwritten digits in grayscale images of size 28x28. The dataset is split into a training set of 69,000 samples and a test set of 1,000 samples. LabelMe (Russell et al. 2008) has 20,019 training images and 2000 test images, each with a 512D GIST descriptor. The descriptors were dimensionality reduced to 40D using PCA. We use this dataset as unsupervised, and the affinity is defined by thresholding in the Euclidean GIST space such that each training point has an average of 100 neighbors.

We evaluate the results by computing a precision-recall graph of varying Hamming thresholds (denoted by α in Equation 2). We compare our methods to the following state-of-the-art spectral hashing methods: Spectral Hashing (SH) (Weiss, Torralba, and Fergus 2009), Anchor Graph Hashing (AGH) (Liu et al. 2011), Inductive Manifold Hashing (IMH) (Shen et al. 2013), Scalable Graph Hashing (SGH) (Jiang and Li 2015), Supervised Discrete Hashing (SDH) (Shen et al. 2015), and Large Graph Hashing with Spectral Rotation (LGHSR) (Li, Hu, and Nie 2017). We use the default settings that the authors provided, and as in Shen et al. (2013) we use settings of anchor number $m = 300$ and neighborhood number $s = 3$. For IMH, we show results using both Laplacian eigenmaps (LE) and t-SNE.

We first compare our method in the unsupervised (or self-supervised) setting to the unsupervised methods listed above. Results are shown in Figure 4a. We show the precision-recall graph of the LabelMe dataset with the self-supervised affinity labels. The results are for the 50 bit code computed for the train set vs. the test set. The results clearly show that our code is more accurate than the other methods over *all* Hamming thresholds.

Next, we compare our method in the supervised scenario. Figure 4b shows the precision-recall of 50 bit codes for the MNIST dataset. The results are computed for the test set only, showing that our method outperforms the other methods



(a) LabelMe (self-supervised) (b) MNIST (c) CIFAR-10

Figure 4: Precision-recall of 50 bit codes with varying Hamming threshold α for different datasets. The methods compared: Spectral Hashing (SH) (Weiss, Torralba, and Fergus 2009), Anchor Graph Hashing (AGH) (Liu et al. 2011), Inductive Manifold Hashing (IMH) (Shen et al. 2013), Scalable Graph Hashing (SGH) (Jiang and Li 2015), Supervised Discrete Hashing (SDH) (Shen et al. 2015), and Large Graph Hashing with Spectral Rotation (LGHSR) (Li, Hu, and Nie 2017).

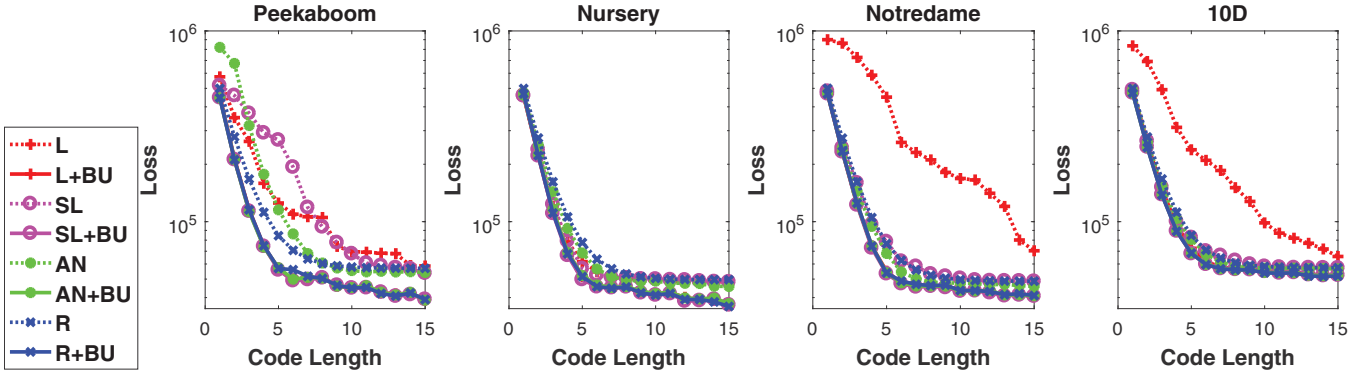


Figure 5: Loss of binary code as a function of code length for PPC with different initial guesses and iterative improvements. This is shown for 4 small datasets. The initial guesses are: signed eigenvector corresponding to the smallest non-trivial eigenvalue of the Laplacian (L) as described in Von Luxburg, signed Laplacian (SL) as suggested by Kunegis, Lommatzsch, and Bauckhage, the sign of the random projection of the 3 smallest non-trivial eigenvalues of the original definition of Laplacian as suggested by Alon and Naor (AN), and random guess (R). We show here results for PPC using only the initial guess, and bit update (BU).

in the more challenging out-of-sample scenario. Similarly, Figure 4c shows the comparison for the CIFAR-10 dataset.

To compare performance at different code lengths, we calculate the area under curve (AUC) for the precision-recall graph in the out-of-sample scenario. Table 1 shows our results on the CIFAR-10 dataset, compared to the results of the spectral methods mentioned above. Our method consistently outperforms the other methods in both short and long codes.

Our solution for the signed min-cut problem includes an iterative scheme that continuously improves the initial guess. As mentioned before, we argue that the initial guess does not play a significant role in the final solution. In fact, at the end of the iterative process, an initial guess based on spectral methods provides similar results to a random initial guess.

In the following experiment, we compute the codes only for the in-sample points (using a fixed random seed) and plot the loss as shown in Equation 4 at each code length. We show our results on the benchmark presented in Norouzi and Blei (2011) for six small datasets, consisting of 1000 training points. Since we use the full versions of the MNIST and LabelMe datasets in the previous sections, we show here the four remaining datasets. We present the results generated from the following initial guesses: signed eigenvector corresponding to the smallest non-trivial eigenvalue of the Laplacian (L) (Von Luxburg 2007), signed Laplacian (SL) (Kunegis, Lommatzsch, and Bauckhage 2009), the sign of the random projection of the 3 smallest non-trivial eigenvalues of the Laplacian (Alon and Naor 2004) (AN), and random guess (R). We show the effect of improving upon the initial guesses using bit update (BU) as presented in Algorithm 2. Results are shown in Figure 5. The results clearly show that the random guess with bit update performs as well as or surpasses the costly spectral computations, while the bit update improves upon all of the initial guesses.

A comparison between vector update (Algorithm 1) and bit update (Algorithm 2) for different initial guesses is shown in Figure 6. It is clear that the bit update method outperforms the vector update. This is reasonable as the bit update is an optimization with smaller steps, allowing for a broader search for the optimum, whereas the vector update takes large steps

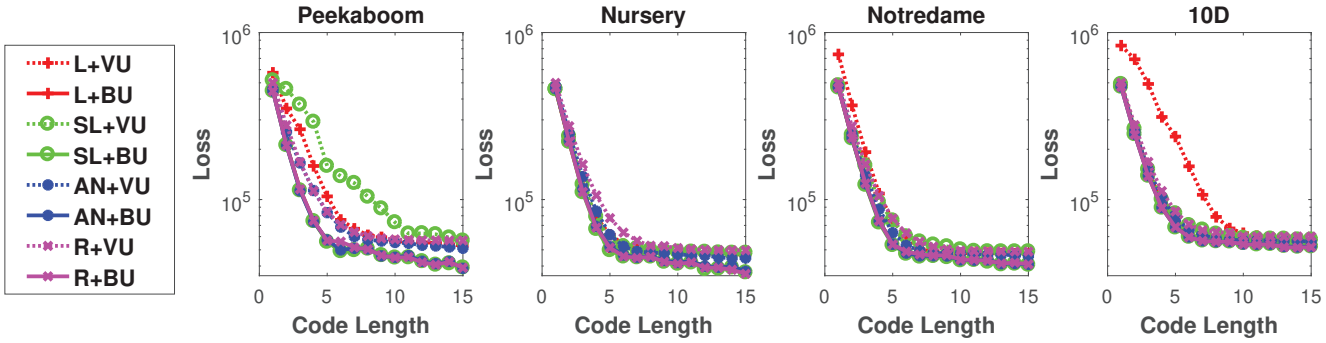


Figure 6: Loss of binary code as a function of code length for PPC with different initial guesses and iterative improvements. This is shown for 4 small datasets. Here we compare the effects of bit update (BU), and vector update (VU). The initial guesses are: signed eigenvector corresponding to the smallest non-trivial eigenvalue of the Laplacian (L) as described in Von Luxburg, signed Laplacian (SL) as suggested by Kunegis, Lommatzsch, and Bauckhage, the sign of the random projection of the 3 smallest non-trivial eigenvalues of the original definition of Laplacian as suggested by Alon and Naor (AN), and random guess (R).

and converges quickly into a local optimum.

Conclusions

We have shown a binary hashing method called Proximity Preserving Code (PPC) based on the signed graph-cut problem. We propose an approximation to this problem and show its advantages over other methods suggested in the literature. We also introduce a hashing framework that can work for both supervised and unsupervised datasets. The framework computes binary code that is more accurate than state-of-the-art graph hashing algorithms, especially in the challenging out-of-sample scenario. We believe the use of the signed graph problem instead of relaxation to the standard graph problem can prove beneficial in other algorithms as well.

Appendix

The following four theorems are used in the Vector Update method in Algorithm 1. The proofs are provided below.

Theorem 1. *for any $n \times n$ matrix W and $\mathbf{b}, \mathbf{b}' \in \{\pm 1\}^n$, if $\mathbf{b}' = sign(W\mathbf{b})$, then $\mathbf{b}'^T W\mathbf{b} \geq \mathbf{b}^T W\mathbf{b}$*

Proof. Denote $\mathbf{u} = W\mathbf{b}$. Thus $\mathbf{b}'^T W\mathbf{b} = \mathbf{b}'^T\mathbf{u} = \sum_i \mathbf{b}'[i]\mathbf{u}[i]$. Since $\mathbf{b}'[i] \in \pm 1$, the maximal value is obtained when $\mathbf{b}'[i] = sign(\mathbf{u}[i])$. We arrive at $\mathbf{b}'^T W\mathbf{b} \geq \mathbf{b}^T W\mathbf{b}$. □

Theorem 2. *Assuming W is positive semidefinite (PSD), if $\mathbf{b}'^T W\mathbf{b} > \mathbf{b}^T W\mathbf{b}$, then $\mathbf{b}'^T W\mathbf{b}' > \mathbf{b}^T W\mathbf{b}$.*

Proof. We express vectors $\mathbf{b}, \mathbf{b}'$ using the eigenvectors of W, $W\mathbf{u}_i = \lambda_i\mathbf{u}_i$, $\mathbf{b} = \sum \alpha_i\mathbf{u}_i$ and $\mathbf{b}' = \sum_i \alpha'_i\mathbf{u}_i$. Given

$$\mathbf{b}'^T W\mathbf{b} = (\sum_i \alpha'_i\mathbf{u}_i^T)(\sum_i \alpha_i\lambda_i\mathbf{u}_i) = \tag{15}$$

$$= \sum_i \alpha'_i\alpha_i\lambda_i \geq \sum_i \alpha_i^2\lambda_i = \mathbf{b}^T W\mathbf{b} \tag{16}$$

we would like to show that

$$\mathbf{b}'^T W\mathbf{b}' = \sum_i \alpha_i'^2\lambda_i \geq \sum_i \alpha_i^2\lambda_i = \mathbf{b}^T W\mathbf{b}$$

Using the PSD property of W ($\lambda_i \geq 0$), we define

$$\mathbf{v} = \begin{bmatrix} \alpha_1\sqrt{\lambda_1} \\ \vdots \\ \alpha_N\sqrt{\lambda_N} \end{bmatrix} \quad \mathbf{v}' = \begin{bmatrix} \alpha'_1\sqrt{\lambda_1} \\ \vdots \\ \alpha'_N\sqrt{\lambda_N} \end{bmatrix}$$

Starting from our original assumption we have: $\mathbf{v}^T\mathbf{v}' \geq \mathbf{v}^T\mathbf{v}$. Using the triangular inequality we have:

$$\|\mathbf{v}\| \cdot \|\mathbf{v}'\| \geq \mathbf{v}^T\mathbf{v}' \geq \|\mathbf{v}\|^2$$

which follows that $\|\mathbf{v}'\| \geq \|\mathbf{v}\|$ and accordingly

$$\mathbf{v}'^T\mathbf{v}' = \mathbf{b}'^T W\mathbf{b}' \geq \mathbf{b}^T W\mathbf{b} = \mathbf{v}^T\mathbf{v}$$

□

Theorem 3. *A symmetric matrix W can become positive semidefinite by applying $W \leftarrow W + |\lambda|I$ where λ is the smallest eigenvalue of W.*

Proof. According to the Gershgorin Circle Theorem (Gershgorin 1931), for an $n \times n$ matrix W, define $R_i = \sum_{j=1, j\neq i}^{n} |w_{ij}|$. All eigenvalues of W are in at least one of the disks $\{v : |v - w_{ii}| \leq R_i\}$. Therefore, by adding the smallest eigenvalue to w_{ii}, the disks will only contain values greater than or equal to zero. □

Theorem 4. *Adding a constant value to the diagonal of the weight matrix W will not affect the output code computed.*

Proof. In PPC, we optimize the vector $\mathbf{b}$ according to Equation 11. Therefore:

$$\begin{aligned} &argmax_{\mathbf{b}}\mathbf{b}^T(W + |\lambda|I)\mathbf{b} \\ &= argmax_{\mathbf{b}}\mathbf{b}^T W\mathbf{b} + |\lambda|\mathbf{b}^T\mathbf{b} \\ &= argmax_{\mathbf{b}}\mathbf{b}^T W\mathbf{b} + |\lambda|n \\ &= argmax_{\mathbf{b}}\mathbf{b}^T W\mathbf{b} \end{aligned}$$

□

References

Alon, N., and Naor, A. 2004. Approximating the Cut-Norm via Grothendieck's Inequality. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, 72–80. ACM.

Chopra, S.; Hadsell, R.; and LeCun, Y. 2005. Learning a Similarity Metric Discriminatively, with Application to Face Verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, 539–546.

Datar, M.; Immorlica, N.; Indyk, P.; and Mirrokni, V. S. 2004. Locality-sensitive Hashing Scheme Based on P-stable Distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, 253–262. ACM.

Deng, L. 2012. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine* 29(6):141–142.

Gershgorin, S. 1931. Uber die Abgrenzung der Eigenwerte einer Matrix. *Izvestija Akademii Nauk SSSR, Serija Matematika* 7(3):749–754.

He, K.; Wen, F.; and Sun, J. 2013. K-means Hashing: An Affinity-Preserving Quantization Method for Learning Binary Compact Codes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2938–2945.

Jegou, H.; Douze, M.; and Schmid, C. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33(1):117–128.

Jiang, Q.-Y., and Li, W.-J. 2015. Scalable Graph Hashing with Feature Transformation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Knyazev, A. V. 2017. Signed Laplacian for Spectral Clustering Revisited. *arXiv preprint*.

Koch, G.; Zemel, R.; and Salakhutdinov, R. 2015. Siamese Neural Networks for One-shot Image Recognition. In *ICML Deep Learning Workshop*, volume 2.

Krizhevsky, A.; Nair, V.; and Hinton, G. 2014. The CIFAR-10 Dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*.

Kulis, B., and Darrell, T. 2009. Learning to Hash with Binary Reconstructive Embeddings. In *Advances in Neural Information Processing Systems*, 1042–1050.

Kunegis, J.; Lommatzsch, A.; and Bauckhage, C. 2009. The Slashdot Zoo: Mining a Social Network with Negative Edges. In *Proceedings of the 18th International Conference on World Wide Web*, 741–750. ACM.

Li, X.; Hu, D.; and Nie, F. 2017. Large Graph Hashing with Spectral Rotation. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Lin, K.; Yang, H.-F.; Hsiao, J.-H.; and Chen, C.-S. 2015. Deep Learning of Binary Hash Codes for Fast Image Retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 27–35.

Liu, A., and Schisterman, E. F. 2004. Principal Component Analysis. *Encyclopedia of Biopharmaceutical Statistics. New York: Marcel Dekker*.

Liu, W.; Wang, J.; Kumar, S.; and Chang, S.-F. 2011. Hashing with Graphs. In *Proceedings of the 28th International Conference on Machine Learning*, 1–8.

Liu, W.; Mu, C.; Kumar, S.; and Chang, S.-F. 2014. Discrete Graph Hashing. In *Advances in Neural Information Processing Systems*, 3419–3427.

Liu, W.; Wen, Y.; Yu, Z.; Li, M.; Raj, B.; and Song, L. 2017. Sphereface: Deep Hypersphere Embedding for Face Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient Estimation of Word Representations in Vector Space. *1st International Conference on Learning Representations*.

Muja, M., and Lowe, D. G. 2014. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36(11):2227–2240.

Norouzi, M., and Blei, D. M. 2011. Minimal Loss Hashing for Compact Binary Codes. In *Proceedings of the 28th International Conference on Machine Learning*, 353–360.

Oliva, A., and Torralba, A. 2001. Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope. *International Journal of Computer Vision* 42(3):145–175.

Russell, B. C.; Torralba, A.; Murphy, K. P.; and Freeman, W. T. 2008. LabelMe: a Database and Web-Based Tool for Image Annotation. *International Journal of Computer Vision* 77(1-3):157–173.

Scholkopf, B., and Smola, A. J. 2001. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press.

Schroff, F.; Kalenichenko, D.; and Philbin, J. 2015. Facenet: A Unified Embedding for Face Recognition and Clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 815–823.

Shen, F.; Shen, C.; Shi, Q.; Van Den Hengel, A.; and Tang, Z. 2013. Inductive Hashing on Manifolds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1562–1569.

Shen, F.; Shen, C.; Liu, W.; and Tao Shen, H. 2015. Supervised Discrete Hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 37–45.

Taigman, Y.; Yang, M.; Ranzato, M.; and Wolf, L. 2014. Deepface: Closing the Gap to Human-level Performance in Face Verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1701–1708.

Von Luxburg, U. 2007. A Tutorial on Spectral Clustering. *Statistics and Computing* 17(4):395–416.

Weiss, Y.; Torralba, A.; and Fergus, R. 2009. Spectral Hashing. In *Advances in Neural Information Processing Systems*, 1753–1760.

Wen, Y.; Zhang, K.; Li, Z.; and Qiao, Y. 2016. A Discriminative Feature Learning Approach for Deep Face Recognition. In *European Conference on Computer Vision*, 499–515. Springer.