Chapter 1
Fast Binary Embedding for High-Dimensional Data

Felix X. Yu, Yunchao Gong, and Sanjiv Kumar

Abstract Binary embedding of high-dimensional data requires long codes to preserve the discriminative power of the input space. Traditional binary coding methods often suffer from very high computation and storage costs in such a scenario. To address this problem, we propose two solutions which improve over existing approaches. The first method, Bilinear Binary Embedding (BBE), converts high-dimensional data to compact similarity-preserving binary codes using compact bilinear projections. Compared to methods that use unstructured matrices for projection, it improves the time complexity from $\mathcal{O}(d^2)$ to $\mathcal{O}(d^{1.5})$, and the space complexity from $\mathcal{O}(d^2)$ to $\mathcal{O}(d)$ where d is the input dimensionality. The second method, Circulant Binary Embedding (CBE), generates binary codes by projecting the data with a circulant matrix. The circulant structure enables the use of Fast Fourier Transformation to speed up the computation. This further improves the time complexity to $\mathcal{O}(d \log d)$. For both BBE and CBE, we propose to learn the projections in a data-dependent fashion. We show by extensive experiments that the proposed approaches give much better performance than the state-of-the-arts for fixed time, and provides much faster computation with no performance degradation for fixed number of bits. The BBE and CBE methods were previously presented in [5, 38]. In this book chapter, we present the two approaches in an unified framework, covering randomized binary embedding, learning-based binary embedding, and learning with dimension reductions. We also discuss the choice of algorithms.

Felix X. Yu
Columbia University, e-mail: yuxinnan@ee.columbia.edu
Yunchao Gong
Facebook AI Research, e-mail: ycgong@facebook.com
Sanjiv Kumar
Google Research, e-mail: sanjivk@google.com

1.1 Introduction

Embedding input data in binary spaces is becoming popular for efficient retrieval and learning on massive data sets [19, 5, 30, 6, 21, 10, 11, 13, 20, 22, 24]. Moreover, in a large number of application domains such as computer vision, biology and finance, data is typically high-dimensional. Taking image retrieval and classification as an example, it has been shown recently that in order to achieve high accuracy on large-scale datasets, it is advantageous to use very high-dimensional descriptors such as Fisher Vectors (FV) [27, 26, 31], Vector of Locally Aggregated Descriptors (VLAD) [15], Locally Constraint Linear Code (LLC) [34], or a large set of weak attributes [37]. When representing such high dimensional data by binary codes, it has been shown that long codes are required in order to achieve good performance. In fact, the required number of bits is $\mathcal{O}(d)$, where d is the input dimensionality [19, 5, 31].

The goal of binary embedding is to well approximate the input distance as Hamming distance so that efficient learning and retrieval can happen directly in the binary space. It is important to note that another related area called *hashing* is a special case with slightly different goal: creating hash tables such that points that are similar fall in the same (or nearby) bucket with high probability. In fact, even in hashing, if high accuracy is desired, one typically needs to use hundreds of hash tables involving tens of thousands of bits.

Most of the existing linear binary coding approaches generate the binary code by applying a "full" (unstructured) projection matrix, followed by a binarization step. Formally, given a data point, $\mathbf{x} \in \mathbb{R}^d$, the k-bit binary code, $h(\mathbf{x}) \in \{+1, -1\}^k$ is generated simply as

$$h(\mathbf{x}) = \text{sgn}(\mathbf{R}\mathbf{x}), \tag{1.1}$$

where $\mathbf{R} \in \mathbb{R}^{k \times d}$, and $\text{sgn}(\cdot)$ is a binary map which returns element-wise sign[1]. Several techniques have been proposed to generate the projection matrix randomly without taking into account the input data [2, 30]. These methods are very popular due to their simplicity but often fail to give the best performance due to their inability to adapt the codes with respect to the input data. Thus, a number of data-dependent techniques have been proposed with different optimization criteria such as reconstruction error [17], data dissimilarity [23, 36], ranking loss [24], quantization error after PCA [7], and pairwise misclassification [35]. These methods are shown to be effective for learning compact codes for relatively low-dimensional data. However, the $\mathcal{O}(d^2)$ computational and space costs prohibit them from being applied to learning long codes for high-dimensional data. For instance, to generate $\mathcal{O}(d)$-bit binary codes for data with $d \sim$1M, a huge projection matrix will be required needing TBs of memory, which is not practical[2].

In order to overcome the computational challenges for the full projection based methods, we propose two approaches reducing both the computational cost and stor-

[1] A few methods transform the linear projection via a nonlinear map before taking the sign [36, 30].

[2] In principle, one can generate the random entries of the matrix on-the-fly (with fixed seeds) without needing to store the matrix. But this will increase the computational time even further.

age cost. The first method, Bilinear Binary Embedding (BBE), reshapes the input vector $\mathbf{x}$ into a matrix $\mathbf{Z}$, and applies a bilinear projection to get the binary code:

$$h(\mathbf{x}) = \text{vec}(\text{sgn}(\mathbf{R}_1^T \mathbf{Z} \mathbf{R}_2)). \tag{1.2}$$

We use $\text{vec}(\cdot)$ to denote an operator which reshapes a matrix to a vector. It is easy to show that when the shapes of $\mathbf{Z}, \mathbf{R}_1, \mathbf{R}_2$ are $\mathcal{O}(\sqrt{d}) \times \mathcal{O}(\sqrt{d})$, the method has time and space complexity of $\mathcal{O}(d^{1.5})$ and $\mathcal{O}(d)$, respectively. The BBE method was originally presented in [5].

The second method, Circulant Binary Embedding (CBE), is even faster than BBE. This is achieved by imposing a *circulant structure* on the projection matrix $\mathbf{R}$ in (1.1).

$$h(\mathbf{x}) = \text{sgn}(\mathbf{R}\mathbf{x}), \quad \mathbf{R} \text{ is a circulant matrix.} \tag{1.3}$$

This special structure allows us to use Fast Fourier Transformation (FFT) based techniques, which have been extensively used in signal processing. The proposed method further reduces the time complexity to $\mathcal{O}(d \log d)$, enabling efficient binary embedding for very high-dimensional data[3]. The CBE method was originally presented in [38].

Table 1.1 compares the time and space complexity for different methods. This book chapter along with [5, 38] make the following contributions:

- We propose the bilinear binary embedding (BBE), and circulant binary embedding (CBE) methods, which reduce both the computational cost and storage cost of binary embedding for high-dimensional data.
- For both the methods, in addition to randomized versions, we propose to learn the data-dependent projections. This helps to further improve the coding quality by considering the data distributions.
- Extensive experiments show that, compared to the state-of-the-art, the proposed method improves the result dramatically for a fixed time cost, and provides much faster computation with no performance degradation for a fixed number of bits.

1.2 Bilinear Binary Embedding (BBE)

Most high-dimensional descriptors have a natural matrix or tensor structure. For example, a HOG descriptor is a two-dimensional grid of histograms, and this structure has been exploited for object detection [29]. A Fisher Vector [27, 26, 31] can be represented as a $k \times 2l$ matrix, where k is the visual vocabulary size and l is the dimensionality of the local image features (the most common choice is SIFT with

[3] One could in principal use other structured matrices like Hadamard matrix along with a sparse random Gaussian matrix to achieve fast projection as was done in fast Johnson-Lindenstrauss transform[1, 3], but it is still slower than circulant projection and needs more space.

Method	Time	Space	Time (Learning)
Full projection	$\mathcal{O}(d^2)$	$\mathcal{O}(d^2)$	$\mathcal{O}(nd^3)$
BBE	$\mathcal{O}(d^{1.5})$	$\mathcal{O}(d)$	$\mathcal{O}(nd^{1.5})$
CBE	$\mathcal{O}(d \log d)$	$\mathcal{O}(d)$	$\mathcal{O}(nd \log d)$

Table 1.1 Comparison of the proposed methods (BBE and CBE) with the full projection based methods for generating long codes (code dimension k comparable to input dimension d). n is the number of instances used for learning data-dependent projection matrices. The $\mathcal{O}(d^{1.5})$ computational complexity of BBE can be achieved when the input vector is reshaped as a $\sqrt{d} \times \sqrt{d}$ matrix. The $\mathcal{O}(d \log d)$ computational complexity of CBE is achieved by using FFT to speed up the computation.

l=128). VLAD [15], which can be seen as a simplified version of FV, can be represented as a $k \times l$ matrix. Finally, an LLC [34] descriptor with s spatial bins can be represented as a $k \times s$ matrix.

Let $\boldsymbol{x} \in \mathbb{R}^d$ denote our descriptor vector. Based on the structure and interpretation of the descriptor, we reorganize it into a $d_1 \times d_2$ matrix with $d = d_1 d_2$:

$$\boldsymbol{x} \in \mathbb{R}^{d_1 d_2 \times 1} \;\; \mapsto \;\; \mathbf{Z} \in \mathbb{R}^{d_1 \times d_2}. \tag{1.4}$$

We assume that each vector $\boldsymbol{x} \in \mathbb{R}^d$ is zero-centered and has unit norm, as L_2 normalization is a widely used preprocessing step that usually improves performance [28].

We will first introduce a randomized method to obtain d-bit bilinear codes in Section 1.2.1 and then explain how to learn data-dependent codes in Section 1.2.2. Learning of reduced-dimension codes will be discussed in Section 1.2.3.

1.2.1 Randomized Bilinear Binary Embedding (Bilinear-rand)

To convert a descriptor $\boldsymbol{x} \in \mathbb{R}^d$ to a d-dimensional binary string, we first consider the framework of [2, 7] that applies a random rotation $\mathbf{R} \in \mathbb{R}^{d \times d}$ to $\boldsymbol{x}$:

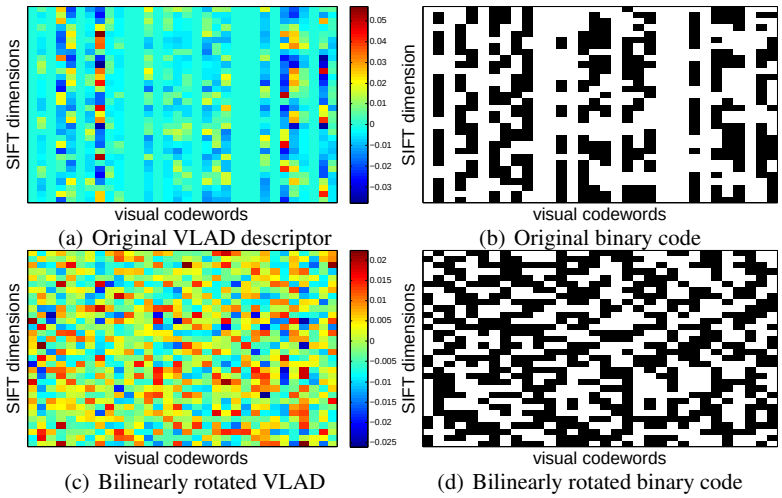
$$h(\boldsymbol{x}) = \operatorname{sgn}(\mathbf{R}\boldsymbol{x}). \tag{1.5}$$

Since $\boldsymbol{x}$ can be represented as a matrix $\mathbf{Z} \in \mathbb{R}^{d_1 \times d_2}$, to make rotation more efficient, we propose a bilinear formulation using two random orthogonal matrices $\mathbf{R}_1 \in \mathbb{R}^{d_1 \times d_1}$ and $\mathbf{R}_2 \in \mathbb{R}^{d_2 \times d_2}$:

$$h(\mathbf{x}) = \operatorname{vec}\Big(\operatorname{sgn}(\mathbf{R}_1^T \mathbf{Z} \mathbf{R}_2)\Big), \tag{1.6}$$

where $\operatorname{vec}(\cdot)$ denotes column-wise concatenation.

It is easy to show that applying a bilinear rotation to $\mathbf{Z} \in \mathbb{R}^{d_1 \times d_2}$ is equivalent to applying a $d_1 d_2 \times d_1 d_2$ rotation to $\operatorname{vec}(\mathbf{Z})$. This rotation is given by $\hat{\mathbf{R}} = \mathbf{R}_2 \otimes \mathbf{R}_1$, where $\otimes$ denotes the Kronecker product:



(a) Original VLAD descriptor (b) Original binary code

(c) Bilinearly rotated VLAD (d) Bilinearly rotated binary code

Fig. 1.1 [5] Visualization of the VLAD descriptor and resulting binary code (given by the sign function) before and after learned bilinear rotation. We only show the first 32 SIFT dimensions and visual codewords. Before the rotation, we can clearly see a block pattern, with many zero values. After the rotation, the descriptor and the binary code look more whitened.

$$\mathrm{vec}(\mathbf{R}_1^T \mathbf{Z} \mathbf{R}_2) = (\mathbf{R}_2^T \otimes \mathbf{R}_1^T)\,\mathrm{vec}(\mathbf{Z}) = \hat{\mathbf{R}}^T\,\mathrm{vec}(\mathbf{Z})$$

follows from the properties of the Kronecker product [18]. Another basic property of the Kronecker product is that if $\mathbf{R}_1$ and $\mathbf{R}_2$ are orthogonal, then $\mathbf{R}_2 \otimes \mathbf{R}_1$ is orthogonal as well [18]. Thus, a bilinear rotation is simply a special case of a full rotation, such that the full rotation matrix $\hat{\mathbf{R}}$ can be reconstructed from two smaller matrices $\mathbf{R}_1$ and $\mathbf{R}_2$.

While the degree of freedom of our bilinear rotation is more restricted than a full rotation, the projection matrices are much smaller, and the projection speed is much faster. In terms of time complexity, performing a full rotation on $\boldsymbol{x}$ takes $\mathcal{O}((d_1 d_2)^2)$ time, while our approach is $\mathcal{O}(d_1^2 d_2 + d_1 d_2^2)$. In terms of space for projections, full rotation takes $\mathcal{O}((d_1 d_2)^2)$, and our approach only takes $\mathcal{O}(d_1^2 + d_2^2)$. For example, for a 64K-dimensional vector, a full rotation will take roughly 16GB of RAM, while the bilinear rotations only take 1MB of RAM. The projection time for a full rotation is more than a second, vs. only 3 ms for bilinear rotations.

Note that when d_1 and d_2 are set as $d_1 = d_2 = d^{1/2}$, the BBE method has the lowest computational complexity $\mathcal{O}(d^{1.5})$, and lowest space complexity $\mathcal{O}(d)$. As computational efficiency is the main focus of this paper, we use such settings in the experiment section. Empirically, tuning d_1 and d_2, or setting them accordingly based on the structure of the descriptor may result in better retrieval performance, but it will lead to higher computational cost.

1.2.2 Learning Bilinear Binary Embedding (Bilinear-opt)

In this section, we present a method for learning the rotations $\mathbf{R}_1$ and $\mathbf{R}_2$ that is inspired by two-sided Procrustes analysis [32] and builds on our earlier work [7, 6].

Following [7], we want to find a rotation $\hat{\mathbf{R}}$ such that the angle θ_i between a rotated feature vector $\hat{\mathbf{R}}^T \boldsymbol{x}_i = \mathrm{vec}(\mathbf{R}_1^T \mathbf{Z}_i \mathbf{R}_2)$ and its binary encoding (geometrically, the nearest vertex of the binary hypercube), $\mathrm{sgn}(\hat{\mathbf{R}}^T \boldsymbol{x}) = \mathrm{vec}(\mathrm{sgn}(\mathbf{R}_1^T \mathbf{Z}_i \mathbf{R}_2))$, is minimized. Given N training points, we want to maximize

$$\begin{aligned}
&\sum\nolimits_{i=1}^{N} \cos(\theta_i) \\
&= \sum\nolimits_{i=1}^{N} \left(\frac{\mathrm{sgn}(\hat{\mathbf{R}}^T \boldsymbol{x}_i)^T}{\sqrt{d}} (\hat{\mathbf{R}}^T \boldsymbol{x}_i) \right) \qquad (1.7) \\
&= \sum\nolimits_{i=1}^{N} \left(\frac{\mathrm{vec}(\mathrm{sgn}(\mathbf{R}_1^T \mathbf{Z}_i \mathbf{R}_2))^T}{\sqrt{d}} \mathrm{vec}\, \mathbf{R}_1^T \mathbf{Z}_i \mathbf{R}_2) \right) \\
&= \frac{1}{\sqrt{d}} \sum\nolimits_{i=1}^{N} \left(\mathrm{vec}(\mathbf{B}_i)^T \mathrm{vec}(\mathbf{R}_1^T \mathbf{Z}_i \mathbf{R}_2) \right) \\
&= \frac{1}{\sqrt{d}} \sum\nolimits_{i=1}^{N} \mathrm{tr}(\mathbf{B}_i \mathbf{R}_2^T \mathbf{Z}_i^T \mathbf{R}_1), \qquad (1.8)
\end{aligned}$$

where $\mathbf{B}_i = \mathrm{sgn}(\mathbf{R}_1^T \mathbf{Z}_i \mathbf{R}_2)$. Notice that (1.7) involves the large projection matrix $\hat{\mathbf{R}} \in \mathbb{R}^{d \times d}$, direct optimization of which is challenging. However, after reformulation into bilinear form (1.8), the expression only involves the two small matrices $\mathbf{R}_1$ and $\mathbf{R}_2$. Letting $\mathcal{B} = \{\mathbf{B}_1, \ldots, \mathbf{B}_N\}$, our objective function is as follows:

$$\begin{aligned}
\mathcal{Q}(\mathcal{B}, \mathbf{R}_1, \mathbf{R}_2) &= \max_{\mathcal{B}, \mathbf{R}_1, \mathbf{R}_2} \sum_{i=1}^{N} \mathrm{tr}(\mathbf{B}_i \mathbf{R}_2^T \mathbf{Z}_i^T \mathbf{R}_1) \qquad (1.9) \\
\text{s.t.} \;\; \mathbf{B}_i &\in \{-1, +1\}^{d_1 \times d_2}, \;\; \mathbf{R}_1^T \mathbf{R}_1 = I, \;\; \mathbf{R}_2^T \mathbf{R}_2 = \mathbf{I}.
\end{aligned}$$

This optimization problem can be solved by block coordinate ascent by alternating between the different variables $\{\mathbf{B}_1, \ldots, \mathbf{B}_N\}$, $\mathbf{R}_1$, and $\mathbf{R}_2$. We describe the update steps for each variable below, assuming the others are fixed.

(S1) Update $\mathbf{B}_i$. When $\mathbf{R}_1$ and $\mathbf{R}_2$ are fixed, we independently solve for each $\mathbf{B}_i$ by maximizing

$$\mathcal{Q}(\mathbf{B}_i) = \mathrm{tr}(\mathbf{B}_i \mathbf{R}_2^T \mathbf{Z}_i^T \mathbf{R}_1) = \textstyle\sum_{k=1}^{d_1} \sum_{l=1}^{d_2} \mathbf{B}_i^{kl} \widetilde{\mathbf{V}}_i^{lk},$$

where $\widetilde{\mathbf{V}}_i^{lk}$ denote the elements of $\widetilde{\mathbf{V}}_i = \mathbf{R}_2^T \mathbf{Z}_i^T \mathbf{R}_1$. $\mathcal{Q}(\mathbf{B}_i)$ is maximized by $\mathbf{B}_i = \mathrm{sgn}(\widetilde{\mathbf{V}}_i^T)$.

(S2) Update $\mathbf{R}_1$. Expanding (1.9) with $\mathbf{R}_2$ and $\mathbf{B}_i$ fixed, we have the following:

$$\begin{aligned}\mathcal{Q}(\mathbf{R}_1) &= \sum\nolimits_{i=1}^{N} \operatorname{tr}(\mathbf{B}_i \mathbf{R}_2^T \mathbf{Z}_i^T \mathbf{R}_1) \\ &= \operatorname{tr}\Big(\sum\nolimits_{i=1}^{N} (\mathbf{B}_i \mathbf{R}_2^T \mathbf{Z}_i^T) \mathbf{R}_1\Big) = \operatorname{tr}(\mathbf{D}_1 \mathbf{R}_1)\,,\end{aligned}$$

where $\mathbf{D}_1 = \sum_{i=1}^{N} (\mathbf{B}_i \mathbf{R}_2^T \mathbf{Z}_i^T)$. The above expression is maximized with the help of polar decomposition: $\mathbf{R}_1 = \mathbf{V}_1 \mathbf{U}_1^T$, where $\mathbf{D}_1 = \mathbf{U}_1 \mathbf{S}_1 \mathbf{V}_1^T$ is the SVD of $\mathbf{D}_1$.
(S3) Update $\mathbf{R}_2$:

$$\begin{aligned}\mathcal{Q}(\mathbf{R}_2) &= \sum\nolimits_{i=1}^{N} \operatorname{tr}(\mathbf{B}_i \mathbf{R}_2^T \mathbf{Z}_i^T \mathbf{R}_1) \\ &= \sum\nolimits_{i=1}^{N} \operatorname{tr}(\mathbf{R}_2^T \mathbf{Z}_i^T \mathbf{R}_1 \mathbf{B}_i) \\ &= \operatorname{tr}\Big(\mathbf{R}_2^T \sum\nolimits_{i=1}^{N} (\mathbf{Z}_i^T \mathbf{R}_1 \mathbf{B}_i)\Big) = \operatorname{tr}(\mathbf{R}_2^T \mathbf{D}_2)\,,\end{aligned}$$

where $\mathbf{D}_2 = \sum_{i=1}^{N} (\mathbf{Z}_i^T \mathbf{R}_1 \mathbf{B}_i)$. Analogously to the update rule for $\mathbf{R}_1$, the update rule for $\mathbf{R}_2$ is $\mathbf{R}_2 = \mathbf{U}_2 \mathbf{V}_2^T$, where $\mathbf{D}_2 = \mathbf{U}_2 \mathbf{S}_2 \mathbf{V}_2^T$ is the SVD of $\mathbf{D}_2$.

We cycle between these updates for several iterations to obtain a local maximum. The convergence of the above program is guaranteed in finite number of iterations as the optimal solution of each step is exactly obtained, each step is guaranteed not to decrease the objective function value, and the objective function is bounded from above. In our implementation, we initialize $\mathbf{R}_1$ and $\mathbf{R}_2$ by random rotations and use three iterations. We have not found significant improvement of performance by using more iterations. The time complexity of this program is $\mathcal{O}(N(d_1^3 + d_2^3))$ where d_1 and d_2 are typically fairly small (e.g., $d_1 = 128,\ d_2 = 500$).

Figure 1.1 visualizes the structure of a VLAD descriptor and the corresponding binary code before and after a learned bilinear rotation.

1.2.3 Learning with Dimensionality Reduction

The formulation of Section 1.2.2 is used to learn d-dimensional binary codes starting from d-dimensional descriptors. Now, to produce a code of size $c = c_1 \times c_2$, where $c_1 < d_1$ and $c_2 < d_2$, we need projection matrices $\mathbf{R}_1 \in \mathbb{R}^{d_1 \times c_1}$, $\mathbf{R}_2 \in \mathbb{R}^{d_2 \times c_2}$ such that $\mathbf{R}_1^T \mathbf{R}_1 = \mathbf{I}$ and $\mathbf{R}_2^T \mathbf{R}_2 = \mathbf{I}$. Each $\mathbf{B}_i$ is now a $c_1 \times c_2$ binary variable. Consider the cosine of the angle between a lower-dimensional projected vector $\hat{\mathbf{R}}^T \boldsymbol{x}_i$ and its binary encoding $\operatorname{sgn}(\hat{\mathbf{R}}^T \boldsymbol{x})$:

$$\cos(\theta_i) = \frac{\operatorname{sgn}(\hat{\mathbf{R}}^T \boldsymbol{x}_i)^T}{\sqrt{c}} \frac{\hat{\mathbf{R}}^T \boldsymbol{x}_i}{\|\hat{\mathbf{R}}^T \boldsymbol{x}_i\|_2}\,,$$

where $\hat{\mathbf{R}} \in \mathbf{R}^{d_1 d_2 \times c_1 c_2}$ and $\hat{\mathbf{R}}^T \hat{\mathbf{R}} = I$. This formulation differs from that of (1.7) in that it contains $\|\hat{\mathbf{R}}^T \boldsymbol{x}_i\|_2$ in the denominator, which makes the optimization difficult [6]. Instead, we follow [6] to define a relaxed objective function based on the

sum of linear correlations

$$\mathcal{Q}(\mathcal{B}, \mathbf{R}_1, \mathbf{R}_2) = \sum\nolimits_{i=1}^{N} \left(\frac{\operatorname{sgn}(\hat{\mathbf{R}}^T \boldsymbol{x}_i)^T}{\sqrt{c}} (\hat{\mathbf{R}}^T \boldsymbol{x}_i) \right).$$

The optimization framework for this objective is similar to that of Section 1.2.2. For the three alternating optimization steps, (S1) remains the same. For (S2) and (S3), we compute the SVD of $\mathbf{D}_1$ and $\mathbf{D}_2$ as $\mathbf{U}_1\mathbf{S}_1\mathbf{V}_1^T$ and $\mathbf{U}_2\mathbf{S}_2\mathbf{V}_2^T$ respectively, and set the two rotations to $\mathbf{R}_1 = \hat{\mathbf{V}}_1\mathbf{U}_1^T$ and $\mathbf{R}_2 = \hat{\mathbf{U}}_2\mathbf{V}_2^T$, where $\hat{\mathbf{V}}_1$ is the top c_1 singular vectors of $\mathbf{V}_1$ and $\hat{\mathbf{U}}_2$ is the top c_2 singular vectors of $\mathbf{U}_2$. To initialize the optimization, we generate random orthogonal directions.

1.3 Circulant Binary Embedding (CBE)

In the former sections, we have proposed the BBE method which can produce binary code with computational complexity $\mathcal{O}(d^{1.5})$. In this section, we propose the circulant binary embedding (CBE) method which is even faster than the BBE method.

A circulant matrix $\mathbf{R} \in \mathbb{R}^{d \times d}$ is a matrix defined by a vector $\mathbf{r} = (r_0, r_1, \cdots, r_{d-1})^T$ [9]. Note that the circulant matrix is sometimes equivalently defined by "circulating" the rows instead of the columns.

$$\mathbf{R} = \operatorname{circ}(\mathbf{r}) := \begin{bmatrix} r_0 & r_{d-1} & \dots & r_2 & r_1 \\ r_1 & r_0 & r_{d-1} & & r_2 \\ \vdots & r_1 & r_0 & \ddots & \vdots \\ r_{d-2} & & \ddots & \ddots & r_{d-1} \\ r_{d-1} & r_{d-2} & \dots & r_1 & r_0 \end{bmatrix}. \tag{1.10}$$

Let $\mathbf{D}$ be a diagonal matrix with each diagonal entry being a Bernoulli variable (± 1 with probability 1/2). For $\mathbf{x} \in \mathbb{R}^d$, its d-bit Circulant Binary Embedding (CBE) with $\mathbf{r} \in \mathbb{R}^d$ is defined as:

$$h(\mathbf{x}) = \operatorname{sgn}(\mathbf{RDx}), \tag{1.11}$$

where $\mathbf{R} = \operatorname{circ}(\mathbf{r})$. The k-bit ($k < d$) CBE is defined as the first k elements of $h(\mathbf{x})$. The need for such a $\mathbf{D}$ is discussed in Section 1.3.1. Note that applying $\mathbf{D}$ to $\mathbf{x}$ is equivalent to applying random sign flipping to each dimension of $\mathbf{x}$. Since sign flipping can be carried out as a preprocessing step for each input $\mathbf{x}$, here onwards for simplicity we will drop explicit mention of $\mathbf{D}$. Hence the binary code is given as $h(\mathbf{x}) = \operatorname{sgn}(\mathbf{Rx})$.

The main advantage of circulant binary embedding is its ability to use Fast Fourier Transformation (FFT) to speed up the computation.

Proposition 1. *For d-dimensional data, CBE has space complexity $\mathcal{O}(d)$, and time complexity $\mathcal{O}(d \log d)$.*

Since a circulant matrix is defined by a single column/row, clearly the storage needed is $\mathcal{O}(d)$. Given a data point $\mathbf{x}$, the d-bit CBE can be efficiently computed as follows. Denote $\circledast$ as operator of circulant convolution. Based on the definition of circulant matrix,

$$\mathbf{R}\mathbf{x} = \mathbf{r} \circledast \mathbf{x}. \tag{1.12}$$

The above can be computed based on Discrete Fourier Transformation (DFT), for which fast algorithm (FFT) is available. The DFT of a vector $\mathbf{t} \in \mathbb{C}^d$ is a d-dimensional vector with each element defined as

$$\mathcal{F}(\mathbf{t})_l = \sum_{m=0}^{d-1} t_n \cdot e^{-i2\pi lm/d}, l = 0, \cdots, d-1. \tag{1.13}$$

The above can be expressed equivalently in a matrix form as

$$\mathcal{F}(\mathbf{t}) = \mathbf{F}_d \mathbf{t}, \tag{1.14}$$

where $\mathbf{F}_d$ is the d-dimensional DFT matrix. Let $\mathbf{F}_d^H$ be the conjugate transpose of $\mathbf{F}_d$. It is easy to show that $\mathbf{F}_d^{-1} = (1/d)\mathbf{F}_d^H$. Similarly, for any $\mathbf{t} \in \mathbb{C}^d$, the Inverse Discrete Fourier Transformation (IDFT) is defined as

$$\mathcal{F}^{-1}(\mathbf{t}) = (1/d)\mathbf{F}_d^H \mathbf{t}. \tag{1.15}$$

Since the convolution of two signals in their original domain is equivalent to the hadamard product in their frequency domain [25],

$$\mathcal{F}(\mathbf{R}\mathbf{x}) = \mathcal{F}(\mathbf{r}) \circ \mathcal{F}(\mathbf{x}). \tag{1.16}$$

Therefore,

$$h(\mathbf{x}) = \operatorname{sgn}\left(\mathcal{F}^{-1}(\mathcal{F}(\mathbf{r}) \circ \mathcal{F}(\mathbf{x}))\right). \tag{1.17}$$

For k-bit CBE, $k < d$, we only need to pick the first k bits of $h(\mathbf{x})$. As DFT and IDFT can be efficiently computed in $\mathcal{O}(d \log d)$ with FFT [25], generating CBE has time complexity $\mathcal{O}(d \log d)$.

1.3.1 Randomized Circulant Binary Embedding (CBE-rand)

A simple way to obtain CBE is by generating the elements of $\mathbf{r}$ in (1.10) independently from the standard normal distribution $\mathcal{N}(0, 1)$. We call this method randomized CBE (CBE-rand). A desirable property of any embedding method is its ability

to approximate input distances in the embedded space. Suppose $\mathcal{H}_k(\mathbf{x}_1, \mathbf{x}_2)$ is the normalized Hamming distance between k-bit codes of a pair of points $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$:

$$\mathcal{H}_k(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{k} \sum_{i=0}^{k-1} \big| \operatorname{sgn}(\mathbf{R}_{i\cdot}\mathbf{x}_1) - \operatorname{sgn}(\mathbf{R}_{i\cdot}\mathbf{x}_2) \big| / 2, \tag{1.18}$$

and $\mathbf{R}_{i\cdot}$ is the i-th row of $\mathbf{R}$, $\mathbf{R} = \operatorname{circ}(\mathbf{r})$. If $\mathbf{r}$ is sampled from $\mathcal{N}(0, 1)$, from [2],

$$\mathbf{Pr}\left(\operatorname{sgn}(\mathbf{r}^T\mathbf{x}_1) \neq \operatorname{sgn}(\mathbf{r}^T\mathbf{x}_2)\right) = \theta/\pi, \tag{1.19}$$

where θ is the angle between $\mathbf{x}_1$ and $\mathbf{x}_2$. Since all the vectors that are circulant variants of $\mathbf{r}$ also follow the same distribution, it is easy to see that

$$\mathbf{E}(\mathcal{H}_k(\mathbf{x}_1, \mathbf{x}_2)) = \theta/\pi. \tag{1.20}$$

For the sake of discussion, if k projections, *i.e.*, first k rows of $\mathbf{R}$, were generated independently, it is easy to show that the variance of $\mathcal{H}_k(\mathbf{x}_1, \mathbf{x}_2)$ will be

$$\mathbf{Var}(\mathcal{H}_k(\mathbf{x}_1, \mathbf{x}_2)) = \theta(\pi - \theta)/k\pi^2. \tag{1.21}$$

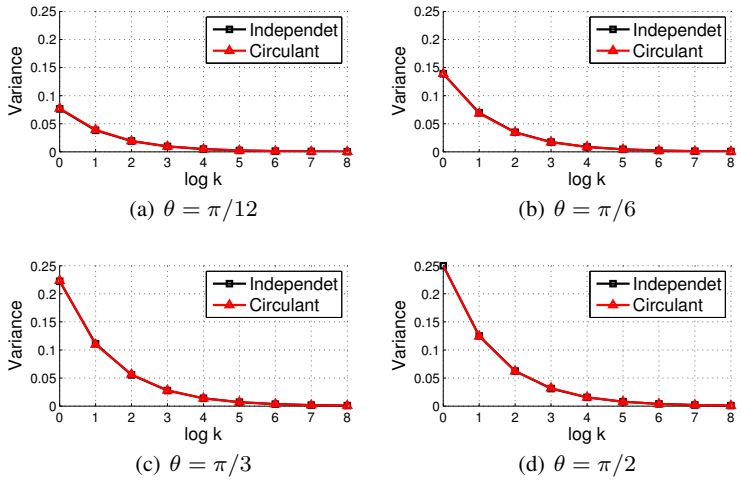
Thus, with more bits (larger k), the normalized hamming distance will be close to the expected value, with lower variance. In other words, the normalized hamming distance approximately preserves the angle[4]. Unfortunately in CBE, the projections are the rows of $\mathbf{R} = \operatorname{circ}(\mathbf{r})$, which are not independent. This makes it hard to derive the variance analytically. To better understand CBE-rand, we run simulations to compare the analytical variance of normalized hamming distance of independent projections (1.21), and the *sample* variance of normalized hamming distance of circulant projections in Figure 1.2. For each θ and k, we randomly generate $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$ such that their angle is θ[5]. We then generate k-dimensional code with CBE-rand, and compute the hamming distance. The variance is estimated by applying CBE-rand 1,000 times. We repeat the whole process 1,000 times, and compute the averaged variance. Surprisingly, the curves of "Independent" and "Circulant" variances are almost indistinguishable. This means that bits generated by CBE-rand are generally as good as the independent bits for angle preservation. An intuitive explanation is that for the circulant matrix, though all the rows are dependent, circulant shifting one or multiple elements will in fact result in very different projections in most cases. We will later show in experiments on real-world data that CBE-rand and Locality Sensitive Hashing (LSH)[6] has almost identical performance (yet CBE-rand is significantly faster) (Section 1.4).

---

[4] In this paper, we consider the case that the data points are ℓ_2 normalized. Therefore the cosine distance, *i.e.*, 1 - $\cos(\theta)$, is equivalent to the l_2 distance.

[5] This can be achieved by extending the 2D points $(1, 0)$, $(\cos\theta, \sin\theta)$ to d-dimension, and performing a random orthonormal rotation, which can be formed by the Gram-Schmidt process on random vectors.

[6] Here, by LSH we imply the binary embedding using $\mathbf{R}$ such that all the rows of $\mathbf{R}$ are sampled iid. With slight abuse of notation, we still call it "hashing" following [2].



(a) $\theta = \pi/12$ (b) $\theta = \pi/6$

(c) $\theta = \pi/3$ (d) $\theta = \pi/2$

Fig. 1.2 [38] The *analytical* variance of normalized hamming distance of independent bits as in (1.21), and the *sample* variance of normalized hamming distance of circulant bits, as a function of angle between points (θ) and number of bits (k). The two curves overlap.

Note that the distortion in input distances after circulant binary embedding comes from two sources: circulant projection, and binarization. For the circulant projection step, recent works have shown that the Johnson-Lindenstrauss-type lemma holds with a slightly worse bound on the number of projections needed to preserve the input distances with high probability [12, 39, 33, 16]. These works also show that before applying the circulant projection, an additional step of randomly flipping the signs of input dimensions is necessary[7]. To show why such a step is required, let us consider the special case when $\mathbf{x}$ is an all-one vector, $\mathbf{1}$. The circulant projection with $\mathbf{R} = \text{circ}(\mathbf{r})$ will result in a vector with all elements to be $\mathbf{r}^T\mathbf{1}$. When $\mathbf{r}$ is independently drawn from $\mathcal{N}(0,1)$, this will be close to 0, and the norm cannot be preserved. Unfortunately the Johnson-Lindenstrauss-type results do not generalize to the distortion caused by the binarization step.

One problem with the randomized CBE method is that it does not utilize the underlying data distribution while generating the matrix $\mathbf{R}$. In the next section, we propose to learn $\mathbf{R}$ in a data-dependent fashion, to minimize the distortions due to circulant projection and binarization.

[7] For each dimension, whether the sign needs to be flipped is predetermined by a $(p = 0.5)$ Bernoulli variable.

1.3.2 Learning Circulant Binary Embedding (CBE-opt)

We propose data-dependent CBE (CBE-opt), by optimizing the projection matrix with a novel time-frequency alternating optimization. We consider the following objective function in learning the d-bit CBE. The extension of learning $k < d$ bits will be shown in Section 1.3.3.

$$\begin{aligned} \underset{\mathbf{B},\mathbf{r}}{\operatorname{argmin}} \quad & ||\mathbf{B} - \mathbf{Z}\mathbf{R}^T||_F^2 + \lambda||\mathbf{R}\mathbf{R}^T - \mathbf{I}||_F^2 \\ \text{s.t.} \quad & \mathbf{R} = \operatorname{circ}(\mathbf{r}), \end{aligned} \tag{1.22}$$

where $\mathbf{Z} \in \mathbb{R}^{n\times d}$, is the data matrix containing n training points: $\mathbf{Z} = [\mathbf{x}_0, \cdots, \mathbf{x}_{n-1}]^T$, and $\mathbf{B} \in \{-1, 1\}^{n\times d}$ is the corresponding binary code matrix.[8]

In the above optimization, the first term minimizes distortion due to binarization. The second term tries to make the projections (rows of $\mathbf{R}$, and hence the corresponding bits) as uncorrelated as possible. In other words, this helps to reduce the redundancy in the learned code. If $\mathbf{R}$ were to be an orthogonal matrix, the second term will vanish and the optimization would find the best rotation such that the distortion due to binarization is minimized. However, when $\mathbf{R}$ is a circulant matrix, $\mathbf{R}$, in general, will not be orthogonal. Similar objective has been used in previous works including [7, 5] and [35].

The above is a combinatorial optimization problem, for which an optimal solution is hard to find. In this section we propose a novel approach to efficiently find a local solution. The idea is to alternatively optimize the objective by fixing $\mathbf{r}$, and $\mathbf{B}$, respectively. For a fixed $\mathbf{r}$, optimizing $\mathbf{B}$ can be easily performed in the input domain ("time" as opposed to "frequency"). For a fixed $\mathbf{B}$, the circulant structure of $\mathbf{R}$ makes it difficult to optimize the objective in the input domain. Hence we propose a novel method, by optimizing $\mathbf{r}$ in the frequency domain based on DFT. This leads to a very efficient procedure.

For a fixed $\mathbf{r}$. The objective is independent on each element of $\mathbf{B}$. Denote B_{ij} as the element of the i-th row and j-th column of $\mathbf{B}$. It is easy to show that $\mathbf{B}$ can be updated as:

$$\begin{aligned} B_{ij} &= \begin{cases} 1 & \text{if } \mathbf{R}_{j\cdot}\mathbf{x}_i \geq 0 \\ -1 & \text{if } \mathbf{R}_{j\cdot}\mathbf{x}_i < 0 \end{cases}, \\ & i = 0, \cdots, n-1. \quad j = 0, \cdots, d-1. \end{aligned} \tag{1.23}$$

For a fixed $\mathbf{B}$. Define $\tilde{\mathbf{r}}$ as the DFT of the circulant vector $\tilde{\mathbf{r}} := \mathcal{F}(\mathbf{r})$. Instead of solving $\mathbf{r}$ directly, we propose to solve $\tilde{\mathbf{r}}$, from which $\mathbf{r}$ can be recovered by IDFT.

Key to our derivation is the fact that DFT projects the signal to a set of orthogonal basis. Therefore the ℓ_2 norm can be preserved. Formally, according to Parseval's theorem , for any $\mathbf{t} \in \mathbb{C}^d$ [25],

[8] If the data is ℓ_2 normalized, we can set $\mathbf{B} \in \{-1/\sqrt{d}, 1/\sqrt{d}\}^{n\times d}$ to make $\mathbf{B}$ and $\mathbf{Z}\mathbf{R}^T$ more comparable. This does not empirically influence the performance.

$$||\mathbf{t}||_2^2 = (1/d)||\mathcal{F}(\mathbf{t})||_2^2.$$

Denote diag$(\cdot)$ as the diagonal matrix formed by a vector. Denote $\Re(\cdot)$ and $\Im(\cdot)$ as the real and imaginary parts, respectively. We use $\mathbf{B}_{i\cdot}$ to denote the i-th row of $\mathbf{B}$. With complex arithmetic, the first term in (1.22) can be expressed in the frequency domain as:

$$\begin{aligned}
&||\mathbf{B} - \mathbf{X}\mathbf{R}^T||_F^2 = \frac{1}{d}\sum_{i=0}^{n-1} ||\mathcal{F}(\mathbf{B}_{i\cdot}^T - \mathbf{R}\mathbf{x}_i)||_2^2 \qquad (1.24)\\
=&\frac{1}{d}\sum_{i=0}^{n-1} ||\mathcal{F}(\mathbf{B}_{i\cdot}^T) - \tilde{\mathbf{r}} \circ \mathcal{F}(\mathbf{x}_i)||_2^2 = \frac{1}{d}\sum_{i=0}^{n-1} ||\mathcal{F}(\mathbf{B}_{i\cdot}^T) - \text{diag}(\mathcal{F}(\mathbf{x}_i))\tilde{\mathbf{r}}||_2^2\\
=&\frac{1}{d}\sum_{i=0}^{n-1} \left(\mathcal{F}(\mathbf{B}_{i\cdot}^T) - \text{diag}(\mathcal{F}(\mathbf{x}_i))\tilde{\mathbf{r}}\right)^T \left(\mathcal{F}(\mathbf{B}_{i\cdot}^T) - \text{diag}(\mathcal{F}(\mathbf{x}_i))\tilde{\mathbf{r}}\right)\\
=&\frac{1}{d}\Big[\Re(\tilde{\mathbf{r}})^T\mathbf{M}\Re(\tilde{\mathbf{r}}) + \Im(\tilde{\mathbf{r}})^T\mathbf{M}\Im(\tilde{\mathbf{r}}) + \Re(\tilde{\mathbf{r}})^T\mathbf{h} + \Im(\tilde{\mathbf{r}})^T\mathbf{g}\Big] + ||\mathbf{B}||_F^2,
\end{aligned}$$

where,

$$\begin{aligned}
\mathbf{M} &= \text{diag}\big(\sum_{i=0}^{n-1} \Re(\mathcal{F}(\mathbf{x}_i)) \circ \Re(\mathcal{F}(\mathbf{x}_i)) + \Im(\mathcal{F}(\mathbf{x}_i)) \circ \Im(\mathcal{F}(\mathbf{x}_i))\big)\\
\mathbf{h} &= -2\sum_{i=0}^{n-1} \Re(\mathcal{F}(\mathbf{x}_i)) \circ \Re(\mathcal{F}(\mathbf{B}_{i\cdot}^T)) + \Im(\mathcal{F}(\mathbf{x}_i)) \circ \Im(\mathcal{F}(\mathbf{B}_{i\cdot}^T))\\
\mathbf{g} &= 2\sum_{i=0}^{n-1} \Im(\mathcal{F}(\mathbf{x}_i)) \circ \Re(\mathcal{F}(\mathbf{B}_{i\cdot}^T)) - \Re(\mathcal{F}(\mathbf{x}_i)) \circ \Im(\mathcal{F}(\mathbf{B}_{i\cdot}^T)).
\end{aligned}$$

The above can be derived based on the following fact. For any $\mathbf{Q} \in \mathbb{C}^{d\times d}$, $\mathbf{s}$, $\mathbf{t} \in \mathbb{C}^d$,

$$\begin{aligned}
&||\mathbf{s} - \mathbf{Q}\mathbf{t}||_2^2 = (\mathbf{s} - \mathbf{Q}\mathbf{t})^H(\mathbf{s} - \mathbf{Q}\mathbf{t}) \qquad (1.25)\\
=&\mathbf{s}^H\mathbf{s} - \mathbf{s}^H\mathbf{Q}\mathbf{t} - \mathbf{t}^H\mathbf{Q}^H\mathbf{s} + \mathbf{t}^H\mathbf{Q}^HA\mathbf{t}\\
=&\Re(\mathbf{s})^T\Re(\mathbf{s}) + \Im(\mathbf{s})^T\Im(\mathbf{s})\\
&- 2\Re(\mathbf{t})^T(\Re(\mathbf{Q})^T\Re(\mathbf{s}) + \Im(\mathbf{Q})^T\Im(\mathbf{s}))\\
&+ 2\Im(\mathbf{t})^T(\Im(\mathbf{Q})^T\Re(\mathbf{s}) - \Re(\mathbf{Q})^T\Im(\mathbf{s}))\\
&+ \Re(\mathbf{t})^T(\Re(\mathbf{Q})^T\Re(\mathbf{Q}) + \Im(\mathbf{Q})^T\Im(\mathbf{Q}))\Re(\mathbf{t})\\
&+ \Im(\mathbf{t})^T(\Re(\mathbf{Q})^T\Re(\mathbf{Q}) + \Im(\mathbf{Q})^T\Im(\mathbf{Q}))\Im(\mathbf{t})\\
&+ 2\Re(\mathbf{t})^T(\Im(\mathbf{Q})^T\Re(\mathbf{Q}) - \Re(\mathbf{Q})^T\Im(\mathbf{Q}))\Im(\mathbf{t}).
\end{aligned}$$

For the second term in (1.22), we note that the circulant matrix can be diagonalized by DFT matrix $\mathbf{F}_d$ and its conjugate transpose $\mathbf{F}_d^H$. Formally, for $\mathbf{R} = \mathrm{circ}(\mathbf{r})$, $\mathbf{r} \in \mathbb{R}^d$,

$$\mathbf{R} = (1/d)\mathbf{F}_d^H \mathrm{diag}(\mathcal{F}(\mathbf{r}))\mathbf{F}_d. \tag{1.26}$$

Let $\mathrm{Tr}(\cdot)$ be the trace of a matrix. Therefore,

$$\begin{aligned}
&||\mathbf{R}\mathbf{R}^T - \mathbf{I}||_F^2 = ||\frac{1}{d}\mathbf{F}_d^H(\mathrm{diag}(\tilde{\mathbf{r}})^H\mathrm{diag}(\tilde{\mathbf{r}}) - \mathbf{I})\mathbf{F}_d||_F^2 \\
=&\,\mathrm{Tr}\left[\frac{1}{d}\mathbf{F}_d^H(\mathrm{diag}(\tilde{\mathbf{r}})^H\mathrm{diag}(\tilde{\mathbf{r}}) - \mathbf{I})^H(\mathrm{diag}(\tilde{\mathbf{r}})^H\mathrm{diag}(\tilde{\mathbf{r}}) - \mathbf{I})\mathbf{F}_d\right] \\
=&\,\mathrm{Tr}\left[(\mathrm{diag}(\tilde{\mathbf{r}})^H\mathrm{diag}(\tilde{\mathbf{r}}) - \mathbf{I})^H(\mathrm{diag}(\tilde{\mathbf{r}})^H\mathrm{diag}(\tilde{\mathbf{r}}) - \mathbf{I})\right] \\
=&||\tilde{\mathbf{r}}^H \circ \tilde{\mathbf{r}} - \mathbf{1}||_2^2 = ||\Re(\tilde{\mathbf{r}})^2 + \Im(\tilde{\mathbf{r}})^2 - \mathbf{1}||_2^2.
\end{aligned} \tag{1.27}$$

Furthermore, as $\mathbf{r}$ is real-valued, additional constraints on $\tilde{\mathbf{r}}$ are needed. For any $u \in \mathbb{C}$, denote $\overline{u}$ as the complex conjugate of u. We have the following result [25]: For any real-valued vector $\mathbf{t} \in \mathbb{C}^d$, $\mathcal{F}(\mathbf{t})_0$ is real-valued, and

$$\mathcal{F}(\mathbf{t})_{d-i} = \overline{\mathcal{F}(\mathbf{t})_i}, \quad i = 1, \cdots, \lfloor d/2 \rfloor.$$

From (1.24) $-$ (1.27), the problem of optimizing $\tilde{\mathbf{r}}$ becomes

$$\begin{aligned}
\underset{\tilde{\mathbf{r}}}{\mathrm{argmin}} \quad & \Re(\tilde{\mathbf{r}})^T\mathbf{M}\Re(\tilde{\mathbf{r}}) + \Im(\tilde{\mathbf{r}})^T\mathbf{M}\Im(\tilde{\mathbf{r}}) + \Re(\tilde{\mathbf{r}})^T\mathbf{h} \\
& + \Im(\tilde{\mathbf{r}})^T\mathbf{g} + \lambda d||\Re(\tilde{\mathbf{r}})^2 + \Im(\tilde{\mathbf{r}})^2 - \mathbf{1}||_2^2 \\
\text{s.t.} \quad & \Im(\tilde{r}_0) = 0 \\
& \Re(\tilde{r}_i) = \Re(\tilde{r}_{d-i}), i = 1, \cdots, \lfloor d/2 \rfloor \\
& \Im(\tilde{r}_i) = -\Im(\tilde{r}_{d-i}), i = 1, \cdots, \lfloor d/2 \rfloor.
\end{aligned} \tag{1.28}$$

The above is non-convex. Fortunately, the objective function can be decomposed, such that we can solve two variables at a time. Denote the diagonal vector of the diagonal matrix $\mathbf{M}$ as $\mathbf{m}$. The above optimization can then be decomposed to the following sets of optimizations.

$$\underset{\tilde{r}_0}{\mathrm{argmin}} \quad m_0\tilde{r}_0^2 + h_0\tilde{r}_0 + \lambda d\left(\tilde{r}_0^2 - 1\right)^2, \text{ s.t. } \tilde{r}_0 = \overline{\tilde{r}_0}. \tag{1.29}$$

$$\begin{aligned}
\underset{\tilde{r}_i}{\mathrm{argmin}} \quad & (m_i + m_{d-i})(\Re(\tilde{r}_i)^2 + \Im(\tilde{r}_i)^2) \\
& + 2\lambda d\left(\Re(\tilde{r}_i)^2 + \Im(\tilde{r}_i)^2 - 1\right)^2 \\
& + (h_i + h_{d-i})\Re(\tilde{r}_i) + (g_i - g_{d-i})\Im(\tilde{r}_i), \\
& i = 1, \cdots, \lfloor d/2 \rfloor.
\end{aligned} \tag{1.30}$$

In (1.29), we need to minimize a 4^{th} order polynomial with one variable, with the closed form solution readily available. In (1.30), we need to minimize a 4^{th} order polynomial with two variables. Though the closed form solution is hard (requiring

d	Full projection	Bilinear projection	Circulant projection
2^{15}	5.44×10^2	2.85	1.11
2^{17}	-	1.91×10^1	4.23
2^{20} (1M)	-	3.76×10^2	3.77×10^1
2^{24}	-	1.22×10^4	8.10×10^2
2^{27} (100M)	-	2.68×10^5	8.15×10^3

Table 1.2 Computational time (ms) of full projection (LSH, ITQ, SH *etc.*), bilinear projection (BBE), and circulant projection (CBE). The time is based on a single 2.9GHz CPU core. The error is within 10%. An empty cell indicates that the memory needed for that method is larger than the machine limit of 24GB.

solution of a cubic bivariate system), we can find local minima by gradient descent, which can be considered as having constant running time for such small-scale problems. The overall objective is guaranteed to be non-increasing in each step. In practice, we can get a good solution with just 5-10 iterations. In summary, the proposed time-frequency alternating optimization procedure has running time $\mathcal{O}(nd \log d)$.

1.3.3 Learning with Dimension Reduction

In the case of learning $k < d$ bits, we need to solve the following optimization problem:

$$\begin{aligned} \underset{\mathbf{B},\mathbf{r}}{\operatorname{argmin}} \quad & ||\mathbf{B}\mathbf{P}_k - \mathbf{X}\mathbf{P}_k^T\mathbf{R}^T||_F^2 + \lambda||\mathbf{R}\mathbf{P}_k\mathbf{P}_k^T\mathbf{R}^T - \mathbf{I}||_F^2 \\ \text{s.t.} \quad & \mathbf{R} = \operatorname{circ}(\mathbf{r}), \end{aligned} \tag{1.31}$$

in which $\mathbf{P}_k = \begin{bmatrix} \mathbf{I}_k & \mathbf{O} \\ \mathbf{O} & \mathbf{O}_{d-k} \end{bmatrix}$, $\mathbf{I}_k$ is a $k \times k$ identity matrix. $\mathbf{O}_{d-k}$ is a $(d-k) \times (d-k)$ all-zero matrix, and $\mathbf{O}$ is a $k \times (d-k)$ all-zero matrix.

In fact, the right multiplication of $\mathbf{P}_k$ can be understood as a "temporal cut-off", which is equivalent to a frequency domain convolution. This makes the optimization difficult, as the objective in frequency domain can no longer be decomposed. To address this issues, we propose a simple solution in which $B_{ij} = 0, i = 0, \cdots, n-1, j = k, \cdots, d-1$ in (1.22). Thus, the optimization procedure remains the same, and the cost is also $\mathcal{O}(nd \log d)$. We will show in experiments that this heuristic provides good performance in practice.

1.4 Experiments

To demonstrate the performance of the proposed binary embedding methods, we conducted experiments on three real-world high-dimensional datasets used by the

current state-of-the-art method for generating binary codes. The Flickr-25600 dataset contains 100K images sampled from a noisy Internet image collection. Each image is represented by a $25,600$ dimensional vector. The ImageNet-51200 contains 100k images sampled from 100 random classes from ImageNet [4], each represented by a $51,200$ dimensional vector. The third dataset (ImageNet-25600) is another random subset of ImageNet containing 100K images in $25,600$ dimensional space. All the vectors are normalized to be of unit norm. Most of the experiment results in this section have been presented in our former work [38].

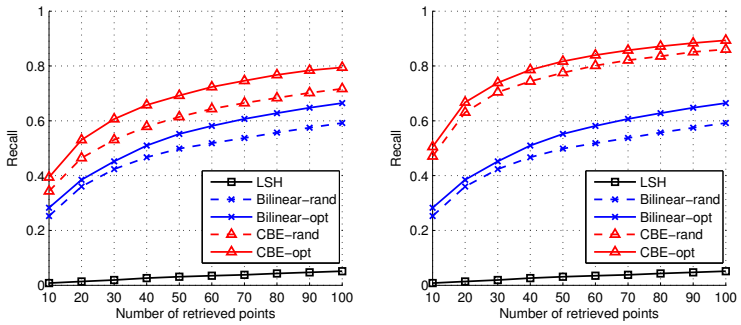
We compared the performance of the randomized (bilinear-rand, CBE-rand) and learned (bilinear-opt, CBE-opt) versions of our embedding methods with the widely used method for high-dimensional data, *i.e.*, LSH. Note that bilinear binary embeddings have been shown to perform similar or better than another promising technique called Product Quantization [14]. We also show an experiment with relatively low-dimensional data in 2048 dimensional space using Flickr data to compare against techniques that perform well for low-dimensional data but do not scale to high-dimensional scenario. Example techniques include ITQ [7], SH [36], SKLSH [30], and AQBC [6].

Following [5, 23, 8], we use 10,000 randomly sampled instances for training. We then randomly sample 500 instances, different from the training set as queries. The performance (recall@1-100) is evaluated by averaging the recalls of the query instances. The ground-truth of each query instance is defined as its 10 nearest neighbors based on ℓ_2 distance. For each dataset, we conduct two sets of experiments: *fixed-time* where code generation time is fixed and *fixed-bits* where the number of bits is fixed across all techniques. We also show an experiment where the binary codes are used for classification. For the bilinear method, in order to get fast computation, the feature vector is reshaped to a near-square matrix, and the dimension of the two bilinear projection matrices are also chosen to be close to square. Parameters for other techniques are tuned to give the best results on these datasets.

Computational Time. When generating k-bit code for d-dimensional data, the full projection, bilinear projection, and circulant projection methods have time complexity $O(kd)$, $O(\sqrt{k}d)$, and $O(d \log d)$, respectively. We compare the computational time in Table 1.2 on a fixed hardware. Based on our implementation, the computational time of the above three methods can be roughly characterized as $d^2 : d\sqrt{d} : 5d \log d$. Note that faster implementation of FFT algorithms will lead to better computational time for CBE by further reducing the constant factor. Due to the small storage requirement $\mathcal{O}(d)$, and the wide availability of highly optimized FFT libraries, CBE is also suitable for implementation on GPU. Our preliminary tests based on GPU shows up to 20 times speedup compared to CPU. In this paper, for fair comparison, we use same CPU based implementation for all the methods.

In addition, the optimizations of learning-based CBE (Section 1.3.2) can be easily solved in a parallel fashion. The small footprints of both BBE and CBE also make them suitable to be implemented on mobile devices, which has strict memory requirement.

Retrieval. The recall for different methods is compared on the three datasets in Figure 1.3, 1.5, and 1.7 shows the performance for different methods when the code



(a) # bits (CBE) = 3,200 (b) # bits (CBE) = 6,400

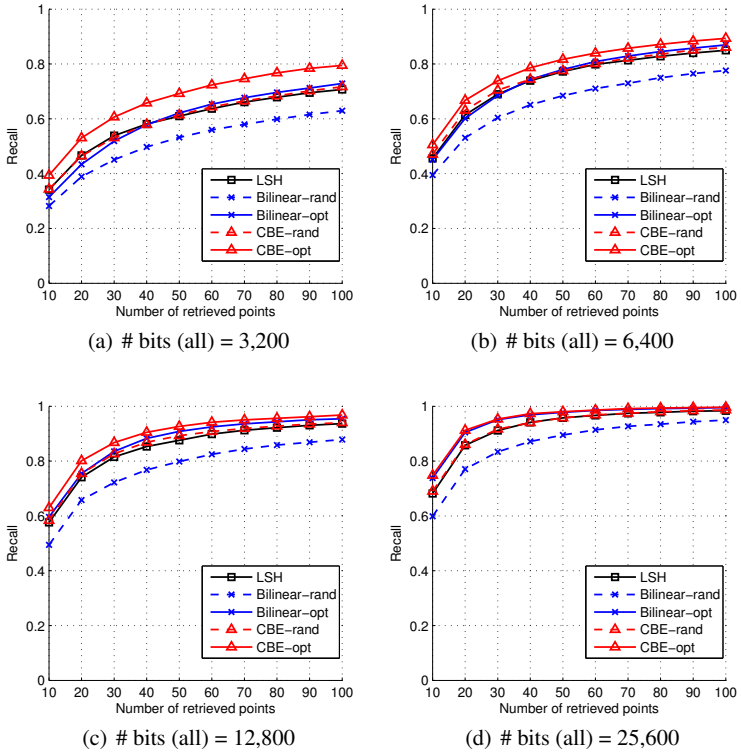
(c) # bits (CBE) = 12,800 (d) # bits (CBE) = 25,600

Fig. 1.3 [38] Recall on Flickr-25600 with fixed time. "# bits" is the number of bits of CBE. Other methods are using less bits to make their computational time identical to CBE. The standard deviation is within 1%.

generation time for all the methods is kept the same as that of CBE. For a fixed time, both CBE and BBE significantly outperform LSH. And CBE outperforms BBE in such high-dimensional settings. Even CBE-rand outperforms LSH and Bilinear code by a large margin.

Figure 1.4, 1.6, and 1.8 compare the performance of different techniques with codes of same length. In this case, the performance of CBE-rand is almost identical to LSH even though it is hundreds of time faster. This is consistent with our analysis in Section 1.3.1. Bilinear-rand is also very competitive to LSH. In addition, CBE-opt/CBE-rand outperform the Bilinear-opt/Bilinear-rand in addition to being 2-3 times faster.

Classification. Besides retrieval, we also test the binary codes for classification. The advantage is to save on storage allowing even large-scale datasets to fit in memory [19, 31]. We follow the asymmetric setting of [31] by training linear SVM on binary code $\mathrm{sgn}(\mathbf{Rx})$, and testing on the original $\mathbf{Rx}$. This empirically has been shown to give better accuracy than the symmetric procedure. We use ImageNet-



(a) # bits (all) = 3,200 (b) # bits (all) = 6,400

(c) # bits (all) = 12,800 (d) # bits (all) = 25,600

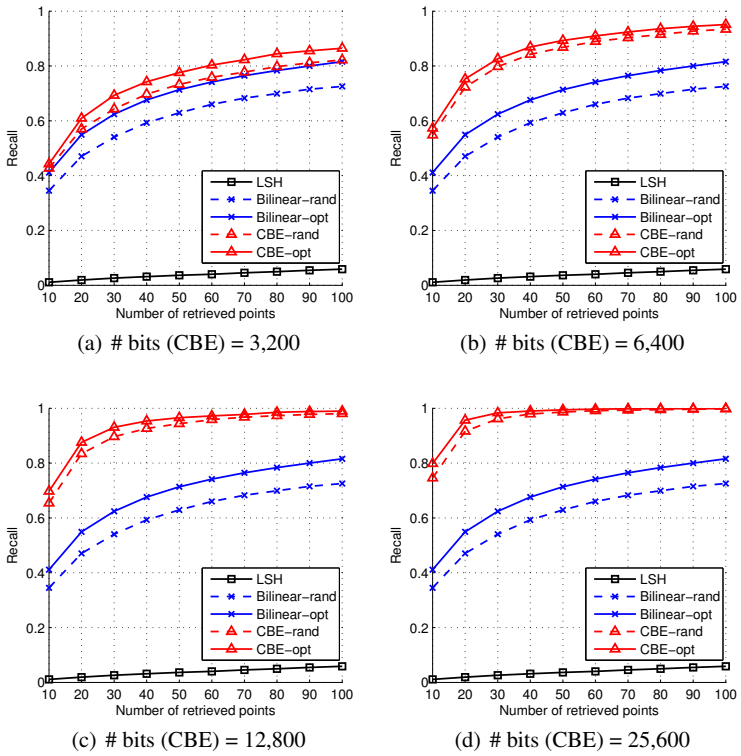
Fig. 1.4 [38] Recall on Flickr-25600 with fixed number of bits. CBE-opt/CBE-rand are 2-3 times faster than Bilinear-opt/Bilinear-rand. Both CBE and BBE(Bilinear) are hundreds of times faster than LSH. The standard deviation is within 1%.

Original	LSH	Bilinear-opt	CBE-opt
25.59±0.33	23.49±0.24	24.02±0.35	24.55 ±0.30

Table 1.3 [38] Multiclass classification accuracy on binary coded ImageNet-25600. The binary codes of same dimensionality are 32 times more space efficient than the original features (single-float).

25600, with randomly sampled 100 images per category for training, 50 for validation and 50 for testing. The code dimension is set as 25,600. As shown in Table 1.3, our methods, which have much faster computation, does not show any performance degradation compared to LSH in classification task as well.

Low-Dimensional Experiment. There exist several techniques that do not scale to high-dimensional case. To compare our method with those, we conducted experiments with fixed number of bits on a relatively low-dimensional dataset (Flickr-2048), constructed by randomly sampling 2,048 dimensions of Flickr-25600. As shown in Figure 1.9, though BBE and CBE are not designed for such scenario,



(a) # bits (CBE) = 3,200 (b) # bits (CBE) = 6,400

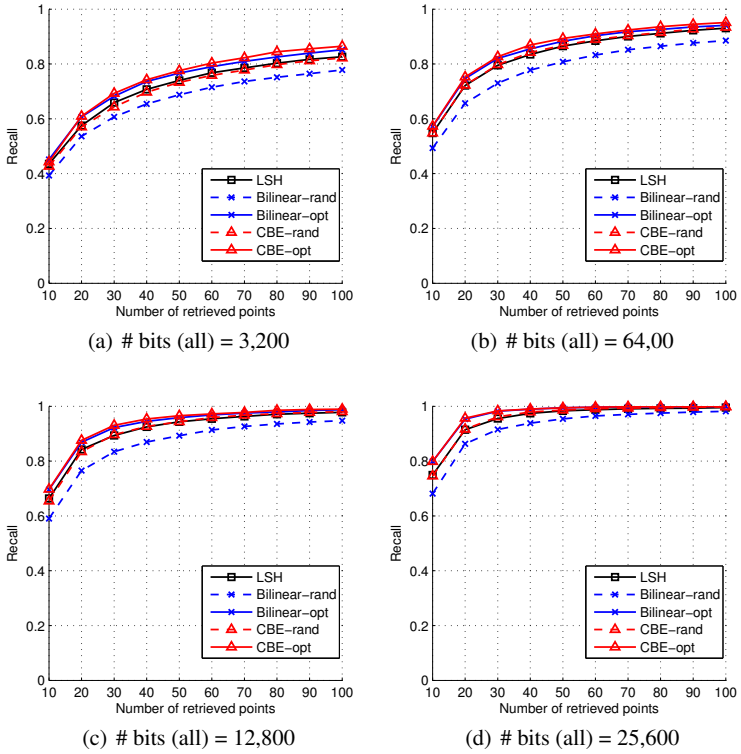
(c) # bits (CBE) = 12,800 (d) # bits (CBE) = 25,600

Fig. 1.5 [38] Recall on ImageNet-25600 with fixed time. "# bits" is the number of bits of CBE. Other methods are using less bits to make their computational time identical to CBE. The standard deviation is within 1%.

they perform better or equivalent to other techniques except ITQ which scales very poorly with d ($\mathcal{O}(d^3)$). Moreover, as the number of bits increases, the gap between ITQ and our methods becomes much smaller suggesting that the performance of ITQ is not expected to be better if one could run ITQ on high-dimensional data. Note that in such small-scale scenario, BBE is faster than CBE due to the computational overhead of FFT.

1.5 Choice of Algorithms

CBE has better computational complexity compared to BBE. In addition, according to the experimental results, with fixed bits, CBE outperforms BBE in general. This suggests that the circulant projection is more powerful than the bilinear projection in



(a) # bits (all) = 3,200 (b) # bits (all) = 64,00

(c) # bits (all) = 12,800 (d) # bits (all) = 25,600

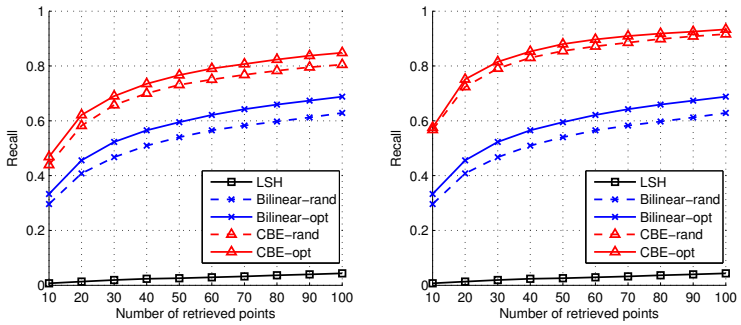
Fig. 1.6 [38] Recall on ImageNet-25600 with fixed number of bits. CBE-opt/CBE-rand are 2-3 times faster than Bilinear-opt/Bilinear-rand. Both CBE and BBE(Bilinear) are hundreds of times faster than LSH. The standard deviation is within 1%.

generating the binary code. This resonates with the works using circulant projections for Johnson-Lindenstrauss transformations [1, 3].

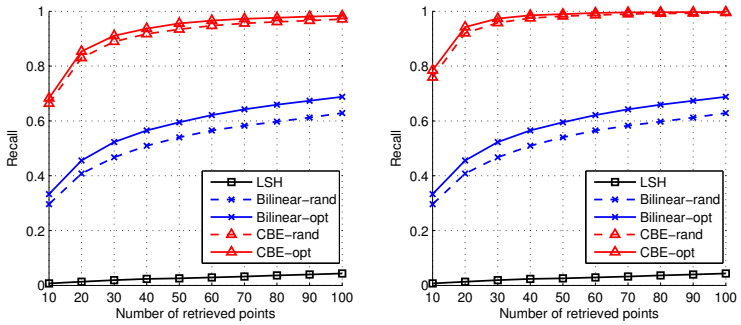
The disadvantage of CBE is the computational overhead of FFT. Based on our implementation, BBE is faster for moderate to high-dimensional data (10k - 30k), and CBE is faster on very high-dimensional data (30k - 100M). Therefore, the final choice of the algorithm should depend on the evaluation metric, and the actual computational cost based on implementation.

1.6 Conclusion

This book chapter introduces two fast binary embedding methods for high-dimensional data [5, 38] with unified notations and framework. The Bilinear Binary Embedding (BBE) has time complexity $\mathcal{O}(d^{1.5})$. The Circulant Binary Embedding (CBE)



(a) # bits (CBE) = 6,400 (b) # bits (CBE) = 12,800

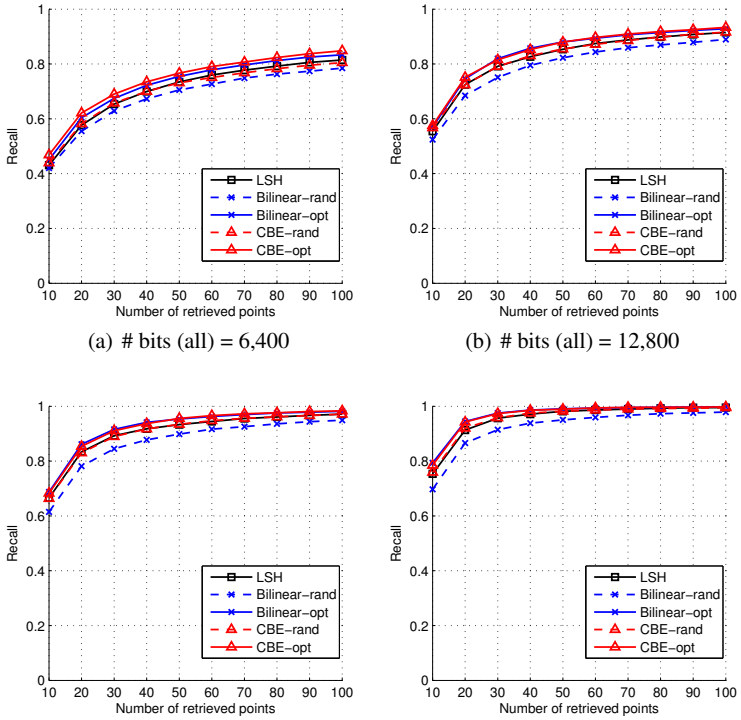


(c) # bits (CBE) = 25,600 (d) # bits (CBE) = 51,200

Fig. 1.7 [38] Recall on ImageNet-51200, with fixed time. "# bits" is the number of bits of CBE. Other methods are using less bits to make their computational time identical to CBE. The standard deviation is within 1%.

has time complexity $\mathcal{O}(d \log d)$. Both algorithms have space complexity $\mathcal{O}(d)$. We have also proposed methods for learning the projection matrices in a data-dependent fashion to further improve the performance. The proposed methods show no performance degradation on real-world data compared to the expensive full projection methods, which has computational complexity $\mathcal{O}(d^2)$. On the contrary, for the fixed time, our methods showed significant accuracy gains.

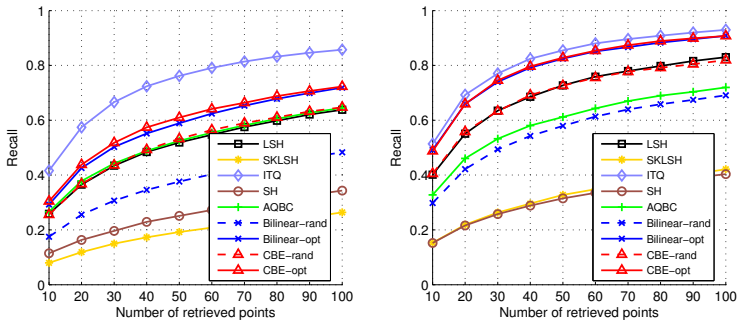
Both the proposed methods use highly structured projections to speed up the computation. Our future work is to study more generalized structured projections for binary embedding. This requires both theoretical analysis on the randomized projections, and novel optimization algorithms for learning data-dependent projections.



(a) # bits (all) = 6,400 (b) # bits (all) = 12,800

(c) # bits (all) = 25,600 (d) # bits (all) = 51,200

Fig. 1.8 [38] Recall on ImageNet-51200 with fixed number of bits. CBE-opt/CBE-rand are 2-3 times faster than Bilinear-opt/Bilinear-rand. Both CBE and BBE(Bilinear) are hundreds of times faster than LSH. The standard deviation is within 1%.



(a) # bits = 1,024 (b) # bits = 2,048

Fig. 1.9 [38] Performance comparison on relatively low-dimensional data (Flickr-2048) with fixed number of bits. CBE gives comparable performance to the state-of-the-art even on low-dimensional data as the number of bits is increased. However, note that these other methods do not scale to very high-dimensional data setting which is the main focus of this work.

References

[1] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *ACM Symposium on Theory of Computing*, 2006.

[2] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *ACM Symposium on Theory of Computing*, 2002.

[3] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. Fast locality-sensitive hashing. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2011.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition*, 2009.

[5] Yunchao Gong, Sanjiv Kumar, Henry A Rowley, and Svetlana Lazebnik. Learning binary codes for high-dimensional data using bilinear projections. In *Computer Vision and Pattern Recognition*, 2013.

[6] Yunchao Gong, Sanjiv Kumar, Vishal Verma, and Svetlana Lazebnik. Angular quantization-based binary codes for fast similarity search. In *Advances in Neural Information Processing Systems*, 2012.

[7] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, 2013.

[8] Albert Gordo and Florent Perronnin. Asymmetric distances for binary embeddings. In *Computer Vision and Pattern Recognition*, 2011.

[9] Robert M Gray. *Toeplitz and circulant matrices: A review*. Now Pub, 2006.

[10] Junfeng He, Regunathan Radhakrishnan, Shih-Fu Chang, and Claus Bauer. Compact hashing with joint optimization of search accuracy and time. In *Computer Vision and Pattern Recognition*, 2011.

[11] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. Spherical hashing. In *Computer Vision and Pattern Recognition*, 2012.

[12] Aicke Hinrichs and Jan Vybíral. Johnson-Lindenstrauss lemma for circulant matrices. *Random Structures & Algorithms*, 39(3):391–398, 2011.

[13] Prateek Jain, Brian Kulis, and Kristen Grauman. Fast image search for learned metrics. In *Computer Vision and Pattern Recognition*, 2008.

[14] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[15] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *Computer Vision and Pattern Recognition*, 2010.

[16] Felix Krahmer and Rachel Ward. New and improved Johnson-Lindenstrauss embeddings via the restricted isometry property. *SIAM Journal on Mathematical Analysis*, 43(3):1269–1281, 2011.

[17] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *Advances in Neural Information Processing Systems*, 2009.

[18] Alan J. Laub. *Matrix Analysis for Scientists and Engineers*. SIAM.

[19] Ping Li, Anshumali Shrivastava, Joshua Moore, and Arnd Christian Konig. Hashing algorithms for large-scale learning. In *Advances in Neural Information Processing Systems*, 2011.

[20] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *Computer Vision and Pattern Recognition*, 2012.

[21] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *International Conference on Machine Learning*, 2011.

[22] Wei Liu, Jun Wang, Yadong Mu, Sanjiv Kumar, and Shih-fu Chang. Compact hyperplane hashing with bilinear functions. In *International Conference on Machine Learning*, 2012.

[23] Mohammad Norouzi and David Fleet. Minimal loss hashing for compact binary codes. In *International Conference on Machine Learning*, 2012.

[24] Mohammad Norouzi, David Fleet, and Ruslan Salakhutdinov. Hamming distance metric learning. In *Advances in Neural Information Processing Systems*, 2012.

[25] Alan V Oppenheim, Ronald W Schafer, John R Buck, et al. *Discrete-time signal processing*, volume 5. Prentice Hall Upper Saddle River, 1999.

[26] Florent Perronnin and Christopher R. Dance. Fisher kernels on visual vocabularies for image categorization. *Computer Vision and Pattern Recognition*, 2007.

[27] Florent Perronnin, Yan Liu, Jorge Sánchez, and Hervé Poirier. Large-scale image retrieval with compressed fisher vectors. In *Computer Vision and Pattern Recognition*, 2010.

[28] Florent Perronnin, J. Sánchez, and Thomas Mensink. Improving the Fisher kernel for large-scale image classification. 2010.

[29] Hamed Pirsiavash, Deva Ramanan, and Charless Fowlkes. Bilinear classifiers for visual recognition. In *Advances in Neural Information Processing Systems*, 2009.

[30] Maxim Raginsky and Svetlana Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Advances in Neural Information Processing Systems*, 2009.

[31] Jorge Sánchez and Florent Perronnin. High-dimensional signature compression for large-scale image classification. In *Computer Vision and Pattern Recognition*, 2011.

[32] PH Schönemann. On two-sided orthogonal Procrustes problems. *Psychometrika*, 1968.

[33] Jan Vybíral. A variant of the Johnson–Lindenstrauss lemma for circulant matrices. *Journal of Functional Analysis*, 260(4):1096–1105, 2011.

[34] Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas Huang, and Yihong Gong. Locality-constrained linear coding for image classification. *Computer Vision and Pattern Recognition*, 2010.

[35] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Sequential projection learning for hashing with compact codes. In *International Conference on Machine Learning*, 2010.

[36] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems*, 2008.

[37] Felix X. Yu, Rongrong Ji, Ming-Hen Tsai, Guangnan Ye, and Shih-Fu Chang. Weak attributes for large-scale image retrieval. In *Computer Vision and Pattern Recognition*, 2012.

[38] Felix X. Yu, Sanjiv Kumar, Yunchao Gong, and Shih-Fu Chang. Circulant binary embedding. In *International Conference on Machine Learning*, 2014.

[39] Hui Zhang and Lizhi Cheng. New bounds for circulant Johnson-Lindenstrauss embeddings. *arXiv preprint arXiv:1308.6339*, 2013.