Int J Comput Vis (2017) 122:439–457
DOI 10.1007/s11263-016-0931-4

Latent Structure Preserving Hashing

Li Liu[1] · **Mengyang Yu**[1] · **Ling Shao**[1]

Received: 14 December 2015 / Accepted: 6 July 2016 / Published online: 20 July 2016

Abstract Aiming at efficient similarity search, hash functions are designed to embed high-dimensional feature descriptors to low-dimensional binary codes such that similar descriptors will lead to binary codes with a short distance in the Hamming space. It is critical to effectively maintain the intrinsic structure and preserve the original information of data in a hashing algorithm. In this paper, we propose a novel hashing algorithm called Latent Structure Preserving Hashing (LSPH), with the target of finding a well-structured low-dimensional data representation from the original high-dimensional data through a novel objective function based on Nonnegative Matrix Factorization (NMF) with their corresponding Kullback-Leibler divergence of data distribution as the regularization term. Via exploiting the joint probabilistic distribution of data, LSPH can automatically learn the latent information and successfully preserve the structure of high-dimensional data. To further achieve robust performance with complex and nonlinear data, in this paper, we also contribute a more generalized multi-layer LSPH (ML-LSPH) framework, in which hierarchical representations can be effectively learned by a multiplicative up-propagation algorithm. Once obtaining the latent representations, the hash functions can be easily acquired through multi-variable logistic regression. Experimental results on three large-scale retrieval datasets, i.e., SIFT 1M, GIST 1M and 500 K TinyImage, show that ML-LSPH can achieve better performance than the single-layer LSPH and both of them outperform existing hashing techniques on large-scale data.

Keywords Hashing · Nonnegative matrix factorization · Latent structure · Dimensionality reduction · Multi-layer extension

Communicated by Xianghua Xie, Mark Jones, Gary Tam.

✉ Ling Shao
ling.shao@ieee.org

Li Liu
li2.liu@northumbria.ac.uk

Mengyang Yu
m.y.yu@ieee.org

[1] Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne NE1 8ST, UK

1 Introduction

Similarity search (Wang et al. 2015; Gionis et al. 1999; Qin et al. 2015; Yu et al. 2015; Liu et al. 2015; Gao et al. 2015; Liu et al. 2015; Zhang et al. 2010; Wang et al. 2014; Bian and Tao 2010) is one of the most critical problems in information retrieval as well as in pattern recognition, data mining and machine learning. Generally speaking, effective similarity search approaches try to construct the index structure in the metric space. However, with the increase of the dimensionality of the data, how to implement the similarity search efficiently and effectively has become an significant topic. To improve retrieval efficiency, hashing algorithms are deployed to find a hash function from Euclidean space to Hamming space. The hashing algorithms with binary coding techniques mainly have two advantages: (1) binary hash codes save storage space; (2) it is efficient to compute the Hamming distance (XOR operation) between the training data and the new coming data in the retrieval procedure of similarity search. The time complexity of searching the hashing table is near $O(1)$.

Current hashing algorithms can be roughly divided into two groups: random projection based hashing and learning based hashing. For the random projection based hashing techniques, the most well-known hashing technique that preserves similarity information is probably Locality-Sensitive

Hashing (LSH) (Gionis et al. 1999). LSH simply employs random linear projections (followed by random thresholding) to map data points close in a Euclidean space to similar codes. It is theoretically guaranteed that as the code length increases, the Hamming distance between two codes will asymptotically approach the Euclidean distance between their corresponding data points. Furthermore, kernelized locality-sensitive hashing (KLSH) (Kulis and Grauman 2009) has also been successfully proposed and utilized for large-scale image retrieval and classification. However, in realistic applications, LSH-related methods usually require long codes to achieve good precision, which result in low recall since the collision probability that two codes fall into the same hash bucket decreases exponentially as the code length increases.

However, the random projection based hash functions are effective only when the binary hash code is long enough. To generate more effective but compact hash codes, a number of methods such as projection learning for hashing have been introduced. Through mining the structure of the data, then being represented on the objective function, a projection learning based hashing algorithm can obtain the hash function by solving an optimization problem associated with the objective function. Spectral Hashing (SpH) (Weiss et al. 2009) is a representative unsupervised hashing method, which can learn compact binary codes that preserve the similarity between documents by forcing the balanced and uncorrelated constraints into the learned codes. Furthermore, principled linear projections like PCA Hashing (PCAH) (Wang et al. 2012) has been suggested for better quantization rather than random projection hashing. Moreover, Semantic Hashing (SH), which is based a stack of Restricted Boltzmann Machines (RBM) (Salakhutdinov and Hinton 2007), was proposed in Salakhutdinov and Hinton (2009). In particular, SH involves two steps: pre-training and fine-tuning. During these two steps, a deep generative model is greedily learned, in which the lowest layer represents the high-dimensional data vector and the highest layer represents the learned binary code for that data. Liu et al. (2011) proposed an Anchor Graph-based Hashing method (AGH), which automatically discovers the neighborhood structure inherent in the data to learn appropriate compact codes. To further make such an approach computationally feasible, the Anchor Graphs used in Liu et al. (2011) were defined with tractable low-rank adjacency matrices. In this way, AGH can allow constant time hashing of a new data point by extrapolating graph Laplacian eigenvectors to eigenfunctions. More recently, Spherical Hashing (SpherH) (Heo et al. 2012) was proposed to map more spatially coherent data points into a binary code compared to hyperplane-based hashing functions. Meanwhile, the authors also developed a new distance function for binary codes, spherical Hamming distance, to achieve final retrieval tasks. Iterative Quantization (ITQ) (Gong et al. 2013) was developed for more compact and effective binary coding. Particularly, a simple and efficient alternating minimization scheme for finding a orthogonal rotation of zero-centered data so as to minimize the quantization error of mapping this data and the vertices of a zero-centered binary hypercube. Additionally, Boosted Similarity Sensitive Coding (BSSC) (Shakhnarovich 2005) was designed to learn a compact and weighted Hamming embedding for task specific similarity search. Boosted binary regression stumps were used as hashing functions to map the input vectors into binary codes. A similar idea as BSSC is also applied to Evolutionary Compact Embedding (ECE) (Liu and Shao 2015), which combines Genetic Programming with the boosting scheme to generate high-quality binary codes for large-scale data classification tasks. Besides, Self-taught Hashing (STH) (Zhang et al. 2010), in which a two-step scheme is effectively applied to learn hash functions, was also successfully utilized for visual retrieval. More hashing techniques can also be seen in Wang et al. (2015), Cao et al. (2012), Song et al. (2014), Song et al. (2013), Liu et al. (2012), Wang et al. (2015), Lin et al. (2013).

Nevertheless, the above mentioned hashing methods have their limitations. Although the random projection based hashing methods, such as LSH, KLSH and SKLSH (Raginsky and Lazebnik 2009), can produce relatively effective codes, such simple linear hash functions cannot reflect the underlying relationship between the data points. Meanwhile, since the long codes are required for acceptable retrieval results via random projection based hashing, the storage space and the cost of computing the Hamming distance will be expensive. On the other hand, in terms of learning based hashing algorithms, most of them, e.g., Shakhnarovich (2005), Weiss et al. (2009), Liu et al. (2011), only focus on the relationship between data or sets rather than considering the combination of the intra-latent structure[1] of data and the inter-probability distribution between the high-dimensional Euclidean space and the low-dimensional Hamming space.

To overcome these limitations above, in this paper, we propose a novel NMF-based approach called Latent Structure Preserving Hashing (LSPH) which can effectively preserve data probabilistic distribution and capture much of the locality structure from the high-dimensional data. In particular, the nonnegative matrix factorization can automatically learn the intra-latent information and the part-based representations of data, while the data probabilistic distribution preserving aims

[1] Usually, one dataset can be composed by a linear/nonlinear combination of a set of latent bases. Each of these bases effectively reflects one or more attributes of the intrinsic data structure. Meanwhile, the intra-latent structure indicates the relationship between the data and these bases. For instance, when we use PCA on face data, the intra-latent structure of a face means the relationship between a face and their Eigenfaces.

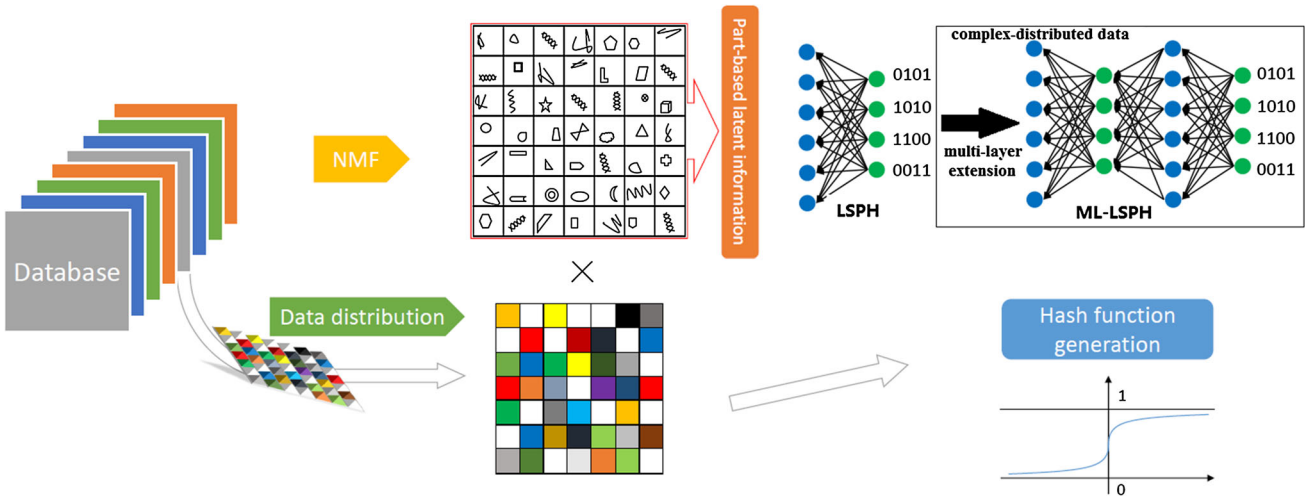

Fig. 1 The outline of the proposed method. Part-based latent information is learned from NMF with the regularization of data distribution. We propose two different versions of our algorithm, i.e., single layer LSPH and multi-layer LSPH. Specifically, ML-LSPH generates deep data representations which can theoretically lead to better performance for retrieval tasks with more complex data

to maintain the similarity between high-dimensional data and low-dimensional codes. Moreover, incorporated with the representation of binary codes, the part-based latent information obtained by NMF based hashing, i.e., LSPH, could be regarded as independent latent attributes of samples. In other words, the binary codes determine whether the high-dimensional data hits the corresponding latent attributes or not. Given an image, this kind of data-driven attributes can allow us to well describe an image and also may benefit zero-shot learning (Jayaraman and Grauman 2014; Lampert et al. 2014; Tao 2015; Yu et al. 2013) for unseen image classification/retrieval in future work.

Specifically, because of the limitation of NMF, which cannot completely discover the latent structure of the original high-dimensional data, we provide a new objective function to preserve as much of the probabilistic distribution structure of the high-dimensional data as possible to the low-dimensional map. Meanwhile, we propose an optimization framework for the objective function and show the updating rules. Besides, to implement the optimization process, the training data are relaxed into a real-valued range. Then, we convert the real-valued representations into binary codes. Finally, we analyze the experimental results and compare them with several existing hashing algorithms. The outline of the proposed LSPH approach is depicted in Fig. 1.

LSPH is a linear hashing technique with a single-layer generative network and data distribution preserving constraints. Although it is an efficient binary coding method for large-scale retrieval tasks, such a single-layer generative network may lead to several limitations in the following cases as mentioned in [1]: (1) when it learns data which lie on or near a nonlinear manifold; (2) when it learns syntactic relationships of given data; and (3) when it learns hierarchically generated data. The single-layer LSPH is apparently not fit for such cases. For instance, LSPH with a single-layer network can well tackle data with small intra-variations such as face images. However, for more complex data with extremely different viewpoints, additional degrees of freedom of the data will be required. In terms of large-scale image retrieval tasks, the sources of data can be very variant and even samples belonging to the same category can differ significantly. Naturally, the single-layer LSPH is not competent for similarity search on such heterogeneous databases.

Therefore, in this paper, we also propose an extension of LSPH called multi-layer LSPH (ML-LSPH) with the multi-layer generative network [1] and distribution preserving constraints. ML-LSPH can deeply learn part-based latent information of data and preserve the joint probabilistic distribution for deep data representations. Applying the sigmoid function to each layer, ML-LSPH is a nonlinear architecture. Similar to recent deep neural networks (Hinton et al. 2006; Masci et al. 2011; Krizhevsky et al. 2012), ML-LSPH generates deep data representations which can theoretically lead to better performance than single layer LSPH for retrieval tasks with more complex data[2] in realistic scenarios. However, ML-LSPH is computationally more expensive during training and test phases compared to single layer LSPH. Thus, there exists a trade-off between ML-LSPH and LSPH in terms of performance and computational complexity, and the choice between these two versions depends on the requirement of the application. Besides, as ML-LSPH is a generalized framework of LSPH, it can easily shrink to LSPH

[2] Such data have large intra-class variations but small inter-class variations, e.g., large-scale retrieval on fine-grained data.

if the number of the layers is set to **1**. We evaluate our LSPH and ML-LSPH on three large-scale datasets: SIFT 1M, GIST 1M and TinyImage, and the results show that our methods significantly outperform the state-of-the-art hashing techniques. It is worthwhile to highlight several contributions of the proposed methods:

- LSPH can learn compact binary codes uncovering the latent semantics and simultaneously preserving the joint probability distribution of data.
- We utilize multivariable logistic regression to generate the hashing function and achieve the out-of-sample extension.
- To tackle the data with more complex distribution, a multi-layer extension of LSPH (i.e., ML-LSPH) has been proposed for large-scale retrieval as well.

The rest of this paper is organized as follows. In Sect. 2, we give a brief review of NMF. The details of LSPH and ML-LSPH are described in Sects. 3 and 4, respectively. Section 5 reports the experimental results. Finally, we conclude this paper and discuss the future work in Sect. 6.

2 A Brief Review of NMF

In this section, we mainly review some related algorithms, focusing on Nonnegative Matrix Factorization (NMF) and its variants. NMF is proposed to learn the nonnegative parts of objects. Given a nonnegative data matrix $X = [\mathbf{x}_1, \cdots, \mathbf{x}_N] \in \mathbb{R}_{\geq 0}^{M \times N}$, each column of X is a sample data. NMF aims to find two nonnegative matrices $U \in \mathbb{R}_{\geq 0}^{M \times D}$ and $V \in \mathbb{R}_{\geq 0}^{D \times N}$ with full rank whose product can approximately represent the original matrix X, i.e., $X \approx UV$. In practice, we always set $D < \min(M, N)$. The target of NMF is to minimize the following objective function

$$\mathcal{L}_{NMF} = \|X - UV\|^2, \text{ s.t. } U, V \geq \mathbf{0}, \tag{1}$$

where $\|\cdot\|$ is the Frobenius norm. To optimize the above objective function, an iterative updating procedure was developed in Lee and Seung (1999) as follows:

$$V_{ij} \leftarrow \frac{\left(U^T X\right)_{ij}}{\left(U^T U V\right)_{ij}} V_{ij}, \; U_{ij} \leftarrow \frac{\left(X V^T\right)_{ij}}{\left(U V V^T\right)_{ij}} U_{ij}, \tag{2}$$

and normalization

$$U_{ij} \leftarrow \frac{U_{ij}}{\sum_i U_{ij}}. \tag{3}$$

It has been proved that the above updating procedure can find the local minimum of $\mathcal{L}_{NMF}$. The matrix V obtained in NMF is always regarded as the low-dimensional representation while the matrix U denotes the basis matrix.

Furthermore, there also exists some variants of NMF. Local NMF (LNMF) (Li et al. 2001) imposes a spatial localized constraint on the bases. In Hoyer (2004), sparse NMF was proposed and later, NMF constrained with neighborhood preserving regularization (NPNMF) (Gu and Zhou 2009) was developed. Besides, researchers also proposed graph regularized NMF (GNMF) (Cai et al. 2011), which effectively preserves the locality structure of data. Beyond these methods, Zhang et al. (2006) extends the original NMF with the kernel trick as kernelized NMF (KNMF), which could extract more useful features hidden in the original data through some kernel-induced nonlinear mappings. Additionally, a hashing method based on multiple kernels NMF was proposed in Liu et al. (2015), where an alternate optimization scheme is applied to determine the combination of different kernels.

In this paper, we present a Latent Structure Preserving NMF framework for hashing (i.e., LSPH), which can effectively preserve the data intrinsic probability distribution and simultaneously reduce the redundancy of low-dimensional representations. Specifically, since the solution of standard NMF only focuses on optimizing matrix factorization to minimize Eq. (1), the obtained low-dimensional representation V lacks the data relationship information. In fact, most of previous NMF extensions are based on keeping the locality regularization to guarantee that, if the high-dimensional data points are close, the low-dimensional representations from NMF can still be close. However, this kind of regularization may lead to a low-quality factorization, since it ignores preserving the whole data distribution but only focuses on locality information. For a realistic scenario with noisy data, locality preserving regularization would even produce worse performance. Rather than locality-based graph regularization, we measure the joint probability of data by Kullback-Leibler divergence, which is defined over all of the potential neighbors and has been proved to effectively resist data noise (Maaten and Hinton 2008). This kind of measurement reveals the global structure such as the presence of clusters at several scales. To make LSPH more capable on data with more complex distributions, the multi-layer LSPH (ML-LSPH) is also proposed, in which more discriminative, high-level representations can be learned from a multi-layer network with the distribution preserving regularization term. To the best of our knowledge, this is the first time that multi-layer NMF based hashing is successfully applied to feature embedding for large-scale similarity search. A preliminary version of our LSPH has been presented in Cai et al. (2015). In this paper, we include more details and experimental results and extend LSPH to ML-LSPH for more complex data in realistic retrieval applications.

3 Latent Structure Preserving Hashing

In this section, we mainly elaborate the proposed Latent Structure Preserving Hashing algorithm.

3.1 Preserving Data Structure with NMF

NMF is an unsupervised learning algorithm which can learn a parts-based representation. Theoretically, it is expected that the low-dimensional data V given by NMF can obtain locality structure from the high-dimensional data X. However, in real-world applications, NMF cannot discover the intrinsic geometrical and discriminating structure of the data space. Therefore, to preserve as much of the significant structure of the high-dimensional data as possible, we propose to minimize the Kullback-Leibler divergence (Xie et al. 2011) between the joint probability distribution in the high-dimensional space and the joint probability distribution that is heavy-tailed in the low-dimensional space:

$$C = \lambda KL(P\|Q). \tag{4}$$

In Eq. (4), P is the joint probability distribution in the high-dimensional space which can also be denoted as p_{ij}. Q is the joint probability distribution in the low-dimensional space that can be represented as q_{ij}. λ is the control of the smoothness of the new representation. The conditional probability p_{ij} means the similarity between data points $\mathbf{x}_i$ and $\mathbf{x}_j$, where $\mathbf{x}_j$ is picked in proportion to their probability density under a Gaussian centered at $\mathbf{x}_i$. Since only significant points are needed to model pairwise similarities, we set p_{ii} and q_{ii} to zero. Meanwhile, it has the characteristics that $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$ for $\forall i, j$. The pairwise similarities in the high-dimensional space p_{ij} are defined as:

$$p_{ij} = \frac{\exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2\right)}{\sum_{k\neq l}\exp\left(-\|\mathbf{x}_k - \mathbf{x}_l\|^2/2\sigma_k^2\right)}, \tag{5}$$

where σ_i is the variance of the Gaussian distribution which is centered on data point x_i. Each data point x_i makes a significant contribution to the cost function. In the low-dimensional map, using the probability distribution that is heavy tailed, the joint probabilities q_{ij} can be defined as:

$$q_{ij} = \frac{\left(1 + \|\mathbf{v}_i - \mathbf{v}_j\|^2\right)^{-1}}{\sum_{k\neq l}\left(1 + \|\mathbf{v}_k - \mathbf{v}_l\|^2\right)^{-1}}. \tag{6}$$

This definition is an infinite mixture of Gaussians, which is much faster to evaluate the density of a point than the single Gaussian, since it does not have an exponential. This representation also makes the mapped points invariant to the changes in the scale for the embedded points that are far apart. Thus, the cost function based on Kullback-Leibler divergence can effectively measure the significance of the data distribution . q_{ij} models p_{ij} is given by

$$G = KL(P\|Q) = \sum_i\sum_j p_{ij}\log p_{ij} - p_{ij}\log q_{ij}. \tag{7}$$

For simplicity, we define two auxiliary variables d_{ij} and Z for making the derivation clearer as follows:

$$d_{ij} = \|\mathbf{v}_i - \mathbf{v}_j\| \text{ and } Z = \sum_{k\neq l}\left(1 + d_{kl}^2\right)^{-1}. \tag{8}$$

Therefore, the gradient of function G with respect to $\mathbf{v}_i$ can be given by

$$\frac{\partial G}{\partial \mathbf{v}_i} = 2\sum_{j=1}^{N}\frac{\partial G}{\partial d_{ij}}\left(\mathbf{v}_i - \mathbf{v}_j\right). \tag{9}$$

Then $\frac{\partial G}{\partial d_{ij}}$ can be calculated by Kullback-Leibler divergence in Eq. (7):

$$\frac{\partial G}{\partial d_{ij}} = -\sum_{k\neq l} p_{kl}\left(\frac{1}{q_{kl}Z}\frac{\partial\left(\left(1+d_{kl}^2\right)^{-1}\right)}{\partial d_{ij}} - \frac{1}{Z}\frac{\partial Z}{\partial d_{ij}}\right). \tag{10}$$

Since $\frac{\partial((1+d_{kl}^2)^{-1})}{\partial d_{ij}}$ is nonzero if and only if $k = i$ and $l = j$, and $\sum_{k\neq l} p_{kl} = 1$, the gradient function can be expressed as

$$\frac{\partial G}{\partial d_{ij}} = 2\left(p_{ij} - q_{ij}\right)\left(1 + d_{ij}^2\right)^{-1}. \tag{11}$$

Eq. (11) can be substituted into Eq. (9). Therefore, the gradient of the Kullback-Leibler divergence between P and Q is

$$\frac{\partial G}{\partial \mathbf{v}_i} = 4\sum_{j=1}^{N}(p_{ij} - q_{ij})(\mathbf{v}_i - \mathbf{v}_j)\left(1 + \|\mathbf{v}_i - \mathbf{v}_j\|^2\right)^{-1}. \tag{12}$$

Therefore, through combining the data structure preserving part in Eq. (4) and the NMF technique, we can obtain the following new objective function:

$$O_f = \|X - UV\|^2 + \lambda KL(P\|Q), \tag{13}$$

where $V \in \{0,1\}^{D\times N}$, $X, U, V \geqslant 0$, $U \in \mathbb{R}^{M\times D}$, $X \in \mathbb{R}^{M\times N}$, and λ controls the smoothness of the new representation.

In most of the circumstances, the low-dimensional data only from NMF is not effective and meaningful for realistic applications. Thus, we introduce $\lambda KL(P\|Q)$ to preserve the structure of the original data which can obtain better results in information retrieval.

3.2 Relaxation and Optimization

Since the discreteness condition $V \in \{0,1\}^{D\times N}$ in Eq. (22) cannot be calculated directly in the optimization procedure, motivated by Weiss et al. (2009), we first relax the data $V \in \{0,1\}^{D\times N}$ to the range $V \in \mathbb{R}^{D\times N}$ for obtaining real-values. Then let the Lagrangian of our problem be:

$$\mathcal{L} = \|X - UV\|^2 + \lambda KL(P\|Q) + tr\left(\Phi U^T\right) + tr(\Psi V^T), \tag{14}$$

where matrices Φ and Ψ are two Lagrangian multiplier matrices. Since we have the gradient of $C = \lambda G$:

$$\frac{\partial C}{\partial \mathbf{v}_i} = 4\lambda \sum_{j=1}^{N} (p_{ij} - q_{ij})(\mathbf{v}_i - \mathbf{v}_j)\left(1 + \|\mathbf{v}_i - \mathbf{v}_j\|^2\right)^{-1}, \tag{15}$$

we make the gradients of $\mathcal{L}$ be zeros to minimize O_f:

$$\frac{\partial \mathcal{L}}{\partial V} = 2\left(-U^T X + U^T UV\right) + \frac{\partial C}{\partial \mathbf{v}_i} + \Psi = \mathbf{0}, \tag{16}$$

$$\frac{\partial \mathcal{L}}{\partial U} = 2\left(-XV^T + UVV^T\right) + \Phi = \mathbf{0}, \tag{17}$$

In addition, we also have KKT conditions: $\Phi_{ij}U_{ij} = 0$ and $\Psi_{ij}V_{ij} = 0, \forall i, j$. Then multiplying V_{ij} and U_{ij} in the corresponding positions on both sides of Eqs. (16) and (17) respectively, we obtain

$$\left(2\left(-U^T X + U^T UV\right) + \frac{\partial C}{\partial \mathbf{v}_i}\right)_{ij} V_{ij} = 0, \tag{18}$$

$$2\left(-XV^T + UVV^T\right)_{ij} U_{ij} = 0. \tag{19}$$

Note that

$$\left(\frac{\partial C}{\partial \mathbf{v}_j}\right)_i = \left(4\lambda \sum_{k=1}^{N} \frac{p_{jk}\mathbf{v}_j - q_{jk}\mathbf{v}_j - p_{jk}\mathbf{v}_k + q_{jk}\mathbf{v}_k}{1 + \|\mathbf{v}_j - \mathbf{v}_k\|^2}\right)_i = 4\lambda \sum_{k=1}^{N} \frac{p_{jk}V_{ij} - q_{jk}V_{ij} - p_{jk}V_{ik} + q_{jk}V_{ik}}{1 + \|\mathbf{v}_j - \mathbf{v}_k\|^2}.$$

Therefore, we have the following update rules for any i, j:

$$V_{ij} \leftarrow \frac{\left(U^T X\right)_{ij} + 2\lambda \sum_{k=1}^{N} \frac{p_{jk}V_{ik} + q_{jk}V_{ij}}{1+\|\mathbf{v}_j - \mathbf{v}_k\|^2}}{\left(U^T UV\right)_{ij} + 2\lambda \sum_{k=1}^{N} \frac{p_{jk}V_{ij} + q_{jk}V_{ik}}{1+\|\mathbf{v}_j - \mathbf{v}_k\|^2}} V_{ij}, \tag{20}$$

$$U_{ij} \leftarrow \frac{\left(XV^T\right)_{ij}}{\left(UVV^T\right)_{ij}} U_{ij}. \tag{21}$$

All the elements in U and V can be guaranteed that they are nonnegative from the allocation. In Lee and Seung (2000), it has been proved that the objective function is monotonically non-increasing after each update of U or V. The proof of convergence about U and V is similar to the ones in Zheng et al. (2011), Cai et al. (2011).

Once the above algorithm is converged, we can obtain the real-valued low-dimensional representation by a linear projection matrix. Since our algorithm is based on general NMF rather than Projective NMF (PNMF) (Yuan and Oja 2005; Guan et al. 2013), a direct projection does not exist for data embedding. Thus, in this paper, inspired by Cai et al. (2007), we consider using linear regression to compute our projection matrix. Particularly, we make the projection orthogonal by solving the Orthogonal Procrustes problem (Schönemann 1966) as follows:

$$\min_{\mathcal{P}} \|\mathcal{P}X - V\|, \text{ s.t. } \mathcal{P}^T\mathcal{P} = I \tag{22}$$

where $\mathcal{P}$ is the orthogonal projection. The optimal solution can be obtained by the following procedure: 1. use the singular value decomposition algorithm to decompose the matrix $X^T V = A\Sigma B^T$; 2. calculate $\mathcal{P} = B\Omega A^T$, where, Ω is a connection matrix as $\Omega = [I, \mathbf{0}] \in \mathbb{R}^{D\times M}$ and $\mathbf{0}$ indicates all zeros matrix. Given data $\mathbf{x} \in \mathbb{R}^{M\times 1}$, its low-dimensional representation is $\mathbf{v} = \mathcal{P}\mathbf{x}$. There are three advantages on using orthogonal projection according to Zhang et al. (2015): Firstly, the orthogonal projection can preserve the Euclidean distance between two points; Secondly, the orthogonal projection can distribute the variance more evenly across the dimensions; Thirdly, the orthogonal projection can learn maximally uncorrelated dimensions, which leads to more compact representations.

3.3 Hash Function Generation

The low-dimensional representations $V \in \mathbb{R}^{D\times N}$ and the bases $U \in \mathbb{R}^{M\times D}$, where $D \ll M$, can be obtained from Eq. (20) and Eq. (21), respectively. Then we need to convert the low-dimensional real-valued representations from $V = [\mathbf{v}_1, \cdots, \mathbf{v}_N]$ into binary codes via thresholding: if the d-th element in $\mathbf{v}_n$ is larger than a specified threshold, this real

value will be represented as 1; otherwise it will be 0, where $d = 1, \cdots, D$ and $n = 1, \cdots, N$.

In addition, a well-designed semantic hashing should also be entropy maximizing to ensure its efficiency (Baluja and Covell 2008). Meanwhile, from the information theory, through having a uniform probability distribution, the source alphabet can reach a maximal entropy. Specifically, if the entropy of codes over the corpus is small, the documents will be mapped to a small number of codes (hash bins). In this paper, the threshold of the elements in $\mathbf{v}_n$ can be set to the median value of $\mathbf{v}_n$, which can satisfy entropy maximization. Therefore, half of the bit-strings will be 1 and the other half will be 0. In this way, the real-value code can be calculated into a binary code (Yu et al. 2014).

However, from the above procedure, we can only obtain the binary codes of the data in the training set. Therefore, given a new sample, it cannot directly find a hash function. To solve such an "out-of-sample" problem, in our approach, we are inspired by the "self-taught" binary coding scheme (Zhang et al. 2010) to use the logistic regression (Hosmer and Lemeshow 2004) which can be treated as a type of probabilistic statistical classification model to compute the hash code for unseen test data. Specifically, we learn a square projection matrix via logistic regression, which can be regarded as a rotation of V. This kind of transformation can make the codes more balanced (Gong et al. 2013; Liu et al. 2012) and lead to better performance compared with directly binarizing V with the median value calculated from training data. To make it more convincing, we also show the performance difference in the later section. Before obtaining the logistic regression cost function, we define that the binary code is represented as $\hat{V} = [\hat{\mathbf{v}}_1, \cdots, \hat{\mathbf{v}}_N]$, where $\hat{\mathbf{v}}_n \in \{0, 1\}^D$ and $n = 1, \cdots, N$. Therefore, the training set can be considered as $\{(\mathbf{v}_1, \hat{\mathbf{v}}_1), (\mathbf{v}_2, \hat{\mathbf{v}}_2), \cdots, (\mathbf{v}_N, \hat{\mathbf{v}}_N)\}$. The vector-valued regression function which is based on the corresponding regression matrix $\Theta \in \mathbb{R}^{D \times D}$ can be represented as

$$h_\Theta(\mathbf{v}_n) = \left(\frac{1}{1 + e^{-(\Theta^T \mathbf{v}_n)_i}} \right)_{i=1,\cdots,D}^T. \tag{23}$$

Therefore, with the maximum log-likelihood criterion for the Bernoulli-distributed data, our cost function for the corresponding regression matrix can be defined as:

$$\begin{aligned} J(\Theta) = -\frac{1}{N}\Big(\sum_{n=1}^{N} \Big(& \hat{\mathbf{v}}_n^T \mathbf{log}(h_\Theta(\mathbf{v}_n)) \\ & + (\mathbf{1} - \hat{\mathbf{v}}_n)^T \mathbf{log}(\mathbf{1} - h_\Theta(\mathbf{v}_n)) \Big) + \delta \|\Theta\|^2 \Big), \end{aligned} \tag{24}$$

where $\mathbf{log}(\cdot)$ is the element-wise logarithm function and $\mathbf{1}$ is an $D \times 1$ all ones matrix. We use $\delta\|\Theta\|^2$ as the regularization term in logistic regression to avoid overfitting.

To find the matrix Θ which aims to minimize $J(\Theta)$, we use gradient descent and repeatedly update each parameter using a learning rate α. The updating equation is shown as follows:

$$\Theta^{(t+1)} = \Theta^{(t)} - \frac{\alpha}{N} \sum_{i=1}^{N} \left(h_{\Theta^{(t)}}(\mathbf{v}_i) - \hat{\mathbf{v}}_i \right) \mathbf{v}_i^T - \frac{\alpha\delta}{N} \Theta^{(t)}. \tag{25}$$

The updating equation stops when the norm of difference between $\Theta^{(t+1)}$ and $\Theta^{(t)}$, i.e., $||\Theta^{(t+1)} - \Theta^{(t)}||^2$, is smaller than a small value. Then we can obtain the regression matrix Θ. For a new coming test data $X_{new} \in \mathbb{R}^{M \times 1}$, then its low-dimensional representation is $V_{new} = \mathcal{P} X_{new}$. Note that each entry of h_Θ is a sigmoid function, the hash codes for a new coming sample $X_{new} \in \mathbb{R}^{M \times 1}$ can be represented as:

$$\hat{V}_{new} = \lfloor h_\Theta(\mathcal{P} X_{new}) \rceil, \tag{26}$$

where $\lfloor \cdot \rceil$ means the nearest integer function for each entry of h_Θ. Specifically, since the output of logistic regression i.e., $h_\Theta(\mathcal{P} X_{new})$, indicates the probability of "1" for each entry, $\lfloor \cdot \rceil$ is equivalent to binarizing each bit by probability 0.5. Thus, if the probability of a bit from $h_\Theta(\mathcal{P} X_{new})$ is larger than 0.5, it will be represented as 1, otherwise 0. For example, through Eq. (26), vector $h_\Theta(\mathcal{P} X_{new}) = [0.17, 0.37, 0.42, 0.79, 0.03, 0.92, \cdots]$ can be expressed as $[0, 0, 0, 1, 0, 1, \cdots]$. Up to now, we can obtain the Latent Structure Preserving Hashing codes for both training and test data. The procedure of LSPH is summarized in Algorithm 1.

Algorithm 1 Latent Structure Preserving Hashing (LSPH)

Input:
The training matrix $X \in \mathbb{R}^{M \times N}$; the objective dimension (code length) D of hash codes; the learning rate α for logistic regression; the regularization parameters $\{\delta, \lambda\}$.

Output:
The basis matrix U, the orthogonal projection $\mathcal{P}$ and the regression matrix Θ.

1: Initialize U and V with uniformly distributed random values between 0 and 1.
2: **repeat**
3: Compute the low-dimensional representation matrix V and the basis matrix U via Eqs. (20) and (21), respectively;
4: **until** convergence
5: SVD decompose the matrix $X^T V$ to obtain $A \Sigma B^T$ and calculate $\mathcal{P} = B \Omega A^T$;
6: Obtain the regression matrix Θ through Eq. (25) and the final LSPH encoding for each sample is defined in Eq. (26).

4 Multi-Layer LSPH Extension

To better tackle the retrieval tasks with more complex data distributions, in this section, we introduce the multi-layer

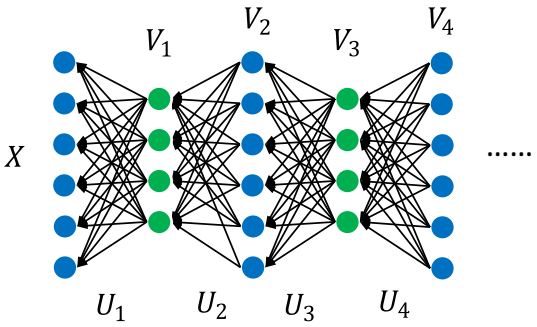

Fig. 2 Illustration of multi-layer LSPH (ML-LSPH)

LSPH (ML-LSPH). ML-LSPH aims to generate more informative high-level representations compared with single-layer LSPH for data with complex distributions. Once the representation by ML-LSPH is computed, the final hashing functions are also obtained through multivariable logistic regression, same as LSPH mentioned above.

Given data matrix $X \in \mathbb{R}^{M\times N}$, inspired by recent deep learning algorithms and multi-layer NMF [1], Trigeorgis et al. (2014), we can extract latent data attributes by incorporating our LSPH algorithm with a multi-layer structure as illustrated in Fig. 2. Similar to the learning of the above representation matrix V, a matrix sequence $V_1, \cdots, V_n$ can be obtained by solving the following optimization problems:

$$\begin{aligned}
&\min \|X - U_1V_1\|^2 + \lambda KL\left(P||Q^{(1)}\right)\\
&\min \|V_1 - U_2V_2\|^2 + \lambda KL\left(P||Q^{(2)}\right)\\
&\qquad\vdots\\
&\min \|V_{n-1} - U_nV_n\|^2 + \lambda KL\left(P||Q^{(n)}\right),
\end{aligned}$$

where $U_i \in \mathbb{R}^{D_{i-1}\times D_i}$, $V_i \in \mathbb{R}^{D_i\times N}$, $i = 1, \cdots, n$, $D_0 = M$, P is the distribution of X and $Q^{(i)}$ is the distribution of V_i.

$$\frac{\partial G_j}{\partial U_i} = U_{i-1}^T \cdots U_{j+1}^T \left(\frac{\partial KL\left(P||Q^{(j)}\right)}{\partial V_j} \odot g'\left(U_{j+1}V_{j+1}\right) \odot \cdots \odot g'\left(U_iV_i\right)\right) V_i^T \tag{27}$$

$$\frac{\partial G_j}{\partial V_i} = U_i^T U_{i-1}^T \cdots U_{j+1}^T \left(\frac{\partial KL\left(P||Q^{(j)}\right)}{\partial V_j} \odot g'\left(U_{j+1}V_{j+1}\right) \odot \cdots \odot g'(U_iV_i)\right) \tag{28}$$

$$(U_i)_{\mu\nu} \leftarrow (U_i)_{\mu\nu} \left(\frac{\left(R_iV_i^T\right)_{\mu\nu} + \lambda\left(\sum_{j=1}^{i}\left(M_{i-1}^{(j)} \odot g'(U_iV_i)\right)V_i^T\right)_{\mu\nu}}{\left(N_iV_i^T\right)_{\mu\nu} + \lambda\left(\sum_{j=1}^{i}\left(S_{i-1}^{(j)} \odot g'(U_iV_i)\right)V_i^T\right)_{\mu\nu}}\right)^{\gamma} \tag{29}$$

$$(V_i)_{\mu\nu} \leftarrow (V_i)_{\mu\nu} \left(\frac{\left(U_i^TR_i\right)_{\mu\nu} + \lambda\left(\sum_{j=1}^{i} U_i^T\left(M_{i-1}^{(j)} \odot g'(U_iV_i)\right)\right)_{\mu\nu}}{\left(U_i^TN_i\right)_{\mu\nu} + \lambda\left(\sum_{j=1}^{i} U_i^T\left(S_{i-1}^{(j)} \odot g'(U_iV_i)\right)\right)_{\mu\nu}}\right)^{\gamma} \tag{30}$$

In this way, $V_i, i = 1, \cdots, n$, are the hidden factors of each layer. By introducing the nonlinear function $g(\cdot)$ into the network, these hidden factors are generated by the following rules:

$$V_i = g\left(U_{i+1}V_{i+1}\right),\ i = n-1, \cdots, 0. \tag{31}$$

Then our task is to minimize the following objective function:

$$F = \|X - g\left(U_1g\left(U_2 \cdots g\left(U_nV_n\right)\right)\right)\|^2 + \lambda\sum_{i=1}^{n} KL\left(P||Q^{(i)}\right). \tag{32}$$

Let us denote $G_i = KL(P||Q^{(i)})$ and $\odot$ is the Hadamard product (element-wise product). By taking the derivatives of F with respect to U_i and V_i, we have:

$$\frac{\partial F}{\partial U_i} = \left(N_i - R_i\right)V_i^T + \lambda\sum_{j=1}^{i}\frac{\partial G_j}{\partial U_i} \tag{33}$$

$$\frac{\partial F}{\partial V_i} = U_i^T\left(N_i - R_i\right) + \lambda\sum_{j=1}^{i}\frac{\partial G_j}{\partial V_i} \tag{34}$$

where matrices $N_i, R_i \in \mathbb{R}^{D_{i-1}\times N}$ are calculated by the following rules:

$$\begin{aligned}
R_{i+1} &= \left(U_i^TR_i\right) \odot g'\left(U_{i+1}V_{i+1}\right)\\
N_{i+1} &= \left(U_i^TN_i\right) \odot g'\left(U_{i+1}V_{i+1}\right)
\end{aligned}$$

for $i = 1, \cdots, n-1$, with the initialization:

$$\begin{aligned}
R_1 &= X \odot g'\left(U_1V_1\right)\\
N_1 &= \left(U_1V_1\right) \odot g'\left(U_1V_1\right).
\end{aligned}$$

Besides, the derivatives of G with respect to U_i and V_i are calculated by Eqs. (27) and (28). With the derivation in Sect. 3, we have the derivative

$$\left(\frac{\partial KL(P||Q^{(j)})}{\partial V_j}\right)_{\mu\nu} = 4\sum_{k=1}^{N}\frac{p_{vk}(V_j)_{\mu\nu} - q_{vk}^{(j)}(V_j)_{\mu\nu} - p_{vk}(V_j)_{\mu k} + q_{vk}^{(j)}(V_j)_{\mu k}}{1 + \|\mathbf{v}_v^j - \mathbf{v}_k^j\|^2},$$

where $\mathbf{v}_k^j$ is the k-th column of $V_j, k = 1, \cdots, N$ and $j = 1, \cdots, n$. To ensure that every element in U_i and V_i is nonnegative, we use the following symbols to split the above derivatives as:

$$\frac{\partial KL\left(P||Q^{(j)}\right)}{\partial V_j} = A_j - B_j, \tag{35}$$

where

$$\left(A_j\right)_{\mu\nu} = 4\sum_{k=1}^{N} \frac{p_{\nu k}(V_j)_{\mu\nu} + q_{\nu k}^{(j)}(V_j)_{\mu k}}{1 + \|\mathbf{v}_\nu^j - \mathbf{v}_k^j\|^2}, \tag{36}$$

$$\left(B_j\right)_{\mu\nu} = 4\sum_{k=1}^{N} \frac{q_{\nu k}^{(j)}(V_j)_{\mu\nu} + p_{\nu k}(V_j)_{\mu k}}{1 + \|\mathbf{v}_\nu^j - \mathbf{v}_k^j\|^2}. \tag{37}$$

Then we can define two matrix sequences S_l and M_l as follows:

$$S_{l+1}^{(j)} = U_{l+1}^T\left(S_l^{(j)} \odot g'(U_{l+1}V_{l+1})\right), \tag{38}$$

$$M_{l+1}^{(j)} = U_{l+1}^T\left(M_l^{(j)} \odot g'(U_{l+1}V_{l+1})\right), \tag{39}$$

where $l = j, \cdots, i-2$, $S_j^{(j)} = A_j$ and $M_j^{(j)} = B_j$. In this way, the derivatives of G_j with respect to U_i and V_i, i.e., Eqs. (27) and (28), will be:

$$\frac{\partial G_j}{\partial U_i} = \left((S_{i-1}^{(j)} - M_{i-1}^{(j)}) \odot g'(U_iV_i)\right)V_i^T, \tag{40}$$

$$\frac{\partial G_j}{\partial V_i} = U_i^T\left((S_{i-1}^{(j)} - M_{i-1}^{(j)}) \odot g'(U_iV_i)\right). \tag{41}$$

Substitute the above equations into Eqs. (33) and (34), we obtain:

$$\begin{aligned}\frac{\partial F}{\partial U_i} =& (N_i - R_i)V_i^T \\ &+ \lambda\sum_{j=1}^{i}\left((S_{i-1}^{(j)} - M_{i-1}^{(j)}) \odot g'(U_iV_i)\right)V_i^T, \\ \frac{\partial F}{\partial V_i} =& U_i^T(N_i - R_i) \\ &+ \lambda\sum_{j=1}^{i} U_i^T\left((S_{i-1}^{(j)} - M_{i-1}^{(j)}) \odot g'(U_iV_i)\right).\end{aligned}$$

Finally, similar to the procedure in Sect. 3, the update rules for multi-layer LSPH (ML-LSPH) are shown in Eqs. (29) and (30), where $0 < \gamma < 1$ is the learning rate and $i = 1, \cdots, n$. The convergence property of the above iteration is similar to the one in [1]. Besides, for better understanding our ML-LSPH, we aim to unify the LSPH and ML-LSPH under a same framework. Thus, in our implementation, the function $g(\cdot)$ applied on U_1 and V_1 is regarded as identity function $f(x) = x$. The function $g(\cdot)$ for U_i and $V_i, i = 2, \cdots, n$ is played by nonlinear sigmoid function $f(x) = \frac{1}{1+e^x}$. In this way, when the number of layers $n = 1$, the ML-LSPH will shrink to the ordinary single-layer LSPH. It is noteworthy that we could theoretically formulate our ML-LSPH to an arbitrary number of layers according the above algorithms. However, for realistic applications with complex data distributions, the number of layers is always less than 3, since when the number of layers increases, the accumulative reconstruction error will cause the non-convergence of the proposed model (Trigeorgis et al. 2014).

For the hash code generating phase, it is similar to LSPH. In particular, acquired the low-dimensional representation V_n in the n-th layer, we first solve the Orthogonal Procrustes problem $\min_{\mathcal{P}^T\mathcal{P}=I} \|\mathcal{P}X - V_n\|$ to achieve the orthogonal projection $\mathcal{P}$. The optimal solution can be obtained by the following procedure: 1. use the singular value decomposition algorithm to decompose the matrix $X^TV_n = A\Sigma B^T$; 2. calculate $\mathcal{P} = B\Omega A^T$, where, Ω is a connection matrix as $\Omega = [I, \mathbf{0}] \in \mathbb{R}^{D\times M}$ and $\mathbf{0}$ indicates all zeros matrix. For a new coming test data $\mathbf{x}_{new} \in \mathbb{R}^{M\times 1}$, the low-dimensional representation in the n-th layer is $\mathbf{v}_n^{new} = \mathcal{P}\mathbf{x}_{new}$ and the binary codes are calculated by $\hat{\mathbf{v}}_n^{new} = \lfloor h_\Theta(\mathcal{P}\mathbf{x}_{new})\rceil$ where Θ is obtained by the similar multi-variable regression scheme. The procedure of ML-LSPH is summarized in Algorithm 2.

Algorithm 2 Multi-layer Latent Structure Preserving Hashing (ML-LSPH)

Input:
The training matrix $X \in \mathbb{R}^{M\times N}$; the dimensions for n layers $D_1, \cdots, D_n$; the learning rate γ for the multi-layer NMF structure; the learning rate α for logistic regression; the regularization parameters $\{\delta, \lambda\}$.

Output:
The n-th layer representation V_n, the orthogonal projection $\mathcal{P}$ and the regression matrix Θ.

1: Initialize U_i and V_i with uniformly distributed random values between 0 and 1, $i = 1, \cdots, n$.
2: **repeat**
3: Compute the basis matrix U_i and the low-dimensional representation matrix V_i via Eqs. (29) and (30) respectively for each layer;
4: **until** convergence
5: SVD decomposes the matrix X^TV_n to obtain $A\Sigma B^T$ and calculate $\mathcal{P} = B\Omega A^T$;
6: Obtain the regression matrix Θ through Eq. (25) and the final ML-LSPH encoding for each sample is defined by $\lfloor h_\Theta(\mathcal{P}X)\rceil$.

Batch-Based Learning Scheme With the number of the layers increasing, the computational costs will inevitably increase as well in the current multi-layer network architecture of ML-LSPH. In order to effectively reduce the computational complexity on large-scale data, we adopt a random batch-based learning strategy (RBLS) in the iteration opti-

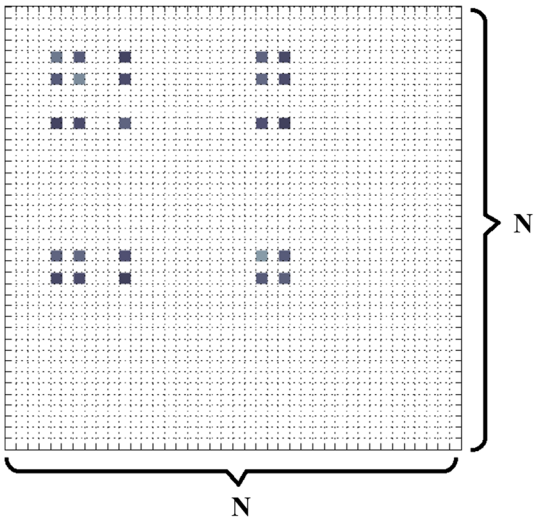

Fig. 3 Illustration of the $P \in \mathbb{R}^{N\times N}$ and $Q^{(j)} \in \mathbb{R}^{N\times N}$ matrices composition. The *white blocks* indicate zeros and *dark-colored blocks* indicate the similarity computed via the randomly selected subset

mization of ML-LSPH. The complexity of each layer's NMF is $O(NMD)$ as mentioned above, which is still regarded as linear complexity in terms of N and not very demanding for large-scale data processing. However, the real bottleneck of the optimization procedure is the calculation of the KL divergence for each layer, specifically, the similarity matrices P and $Q^{(m)}$, due to the complexity of $O(N^2D)$. Therefore, in our implementation, we adopt RBLS to effectively reduce the complexity for computing P and $Q^{(m)}$ in ML-LSPH. In detail, for each step of updating P and $Q^{(m)}$, we randomly select a small subset of the whole training set. Then we only need to compute the pairwise similarity of this subset and the rest of the elements of P and $Q^{(m)}$ are replaced by zeros:

$$p_{ij} = \begin{cases} \frac{\exp(-\|\mathbf{x}_i-\mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq l}\exp(-\|\mathbf{x}_k-\mathbf{x}_l\|^2/2\sigma_k^2)}, & \text{if } \mathbf{x}_i, \mathbf{x}_j \in \text{batch} \\ 0, & \text{otherwise} \end{cases},$$

$$q_{ij}^m = \begin{cases} \frac{(1+\|\mathbf{v}_i^m-\mathbf{v}_j^m\|^2)^{-1}}{\sum_{k\neq l}(1+\|\mathbf{v}_k^m-\mathbf{v}_l^m\|^2)^{-1}}, & \text{if } \mathbf{v}_i^m, \mathbf{v}_j^m \in \text{batch} \\ 0, & \text{otherwise} \end{cases},$$

where m indicates the layer index. The illustration of proposed RBLS are also shown in Fig. 3. If we assume the size of the small subset is ℓ, the complexity of our RBLS will be reduced from $O(N^2D)$ to $O(\ell^2 D)$. Usually, the ℓ can be set as $\ell = 1/100N$. It is noteworthy that only the computation of P and $Q^{(m)}$ are replaced by the above RBLS trick and other parts of the algorithm in ML-LSPH are still the same as mentioned before. In this way, our multi-layer LSPH becomes scalable for large-scale data.

5 Computational Complexity Analysis

In this section, we will discuss the computational complexity of LSPH and ML-LSPH. The computational complexity of LSPH consists of three parts. The first part is for computing NMF, the complexity of which is $O(NMD)$ (Li et al. 2014), where N is the size of the dataset, M and D are the dimensionalities of the high-dimensional data and the low-dimensional data respectively. The second part is to compute the cost function Eq. (7) in the objective function which has the complexity $O(N^2D)$. The last part is the logistic regression procedure whose complexity is $O(ND^2)$. Therefore, the total computational complexity of LSPH is: $O(tNMD + N^2D + tND^2)$, where t is the number of iterations.

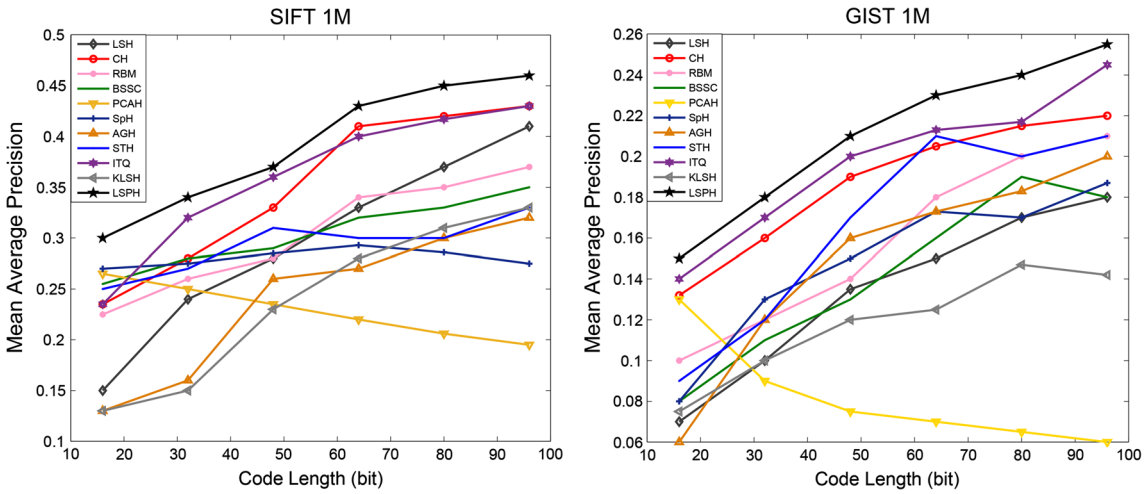

Fig. 4 The Mean Average Precision of the compared algorithms on the SIFT 1M and GIST 1M datasets (Color figure online)

It is obvious that single-layer ML-LSPH is actually LSPH. The computational complexity of ML-LSPH with multiple layers consists of several NMF optimizations, the computation of matrices P and $Q^{(j)}$, and the logistic regression procedure. With the above discussion, the computational complexity of ML-LSPH is $O(tNM\sum_{i=1}^{n} D_i + \ell^2 \sum_{i=1}^{n} D_i + tND^2)$, where ℓ is the batch size.

6 Experiments and Results

In this section, we systematically evaluate the proposed LSPH and multi-layer LSPH (ML-LSPH) on three large-scale datasets. The relevant experimental results and data visualization will be included in the rest of this section. All the experiments are completed using MatLab2014a on a work-

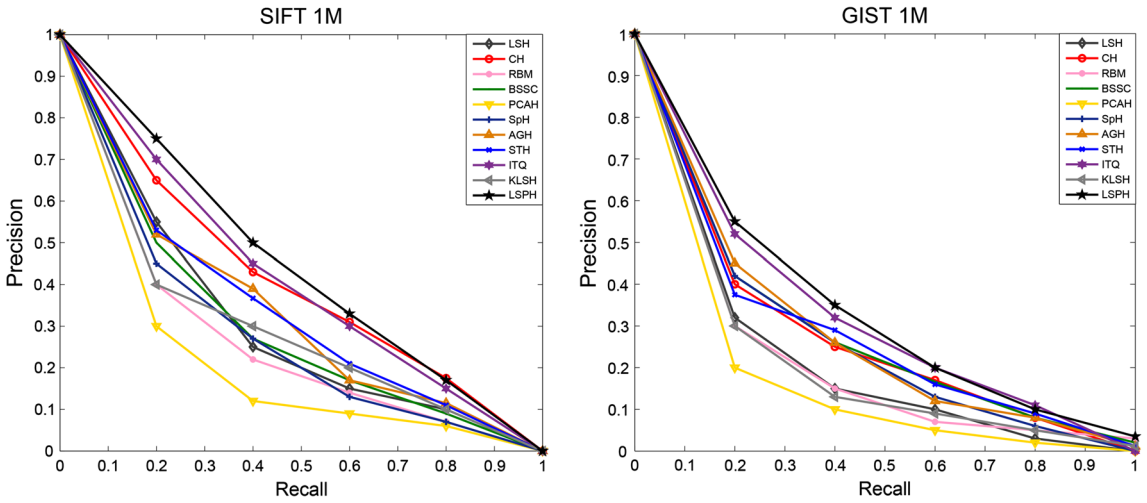

Fig. 5 The precision-recall curves of the compared algorithms on the SIFT 1M and GIST 1M datasets for the code of 48 bits (Color figure online)

Table 1 The comparison of MAP between using median values and Logistic regression to generate codes with different numbers of bits

Code length	16 bits	32 bits	48 bits	64 bits	80 bits	96 bits
SIFT 1M						
Median value binarization	0.274	0.321	0.355	0.390	0.427	0.448
Logistic regression binarization	**0.303**	**0.340**	**0.376**	**0.424**	**0.445**	**0.461**
GIST 1M						
Median value binarization	0.131	0.157	0.193	0.207	0.214	0.227
Logistic regression binarization	**0.153**	**0.185**	**0.211**	**0.234**	**0.241**	**0.255**

Table 2 The comparison of MAP, training time and test time of 32 bits and 48 bits of all the compared algorithms on the SIFT 1M dataset

Methods	SIFT 1M					
	32 bits			48 bits		
	MAP	Train time (s)	Test time (μs)	MAP	Train time (s)	Test time (μs)
LSH	0.240	0.3	1.1	0.280	0.6	1.9
KLSH	0.150	10.5	14.6	0.230	10.7	16.2
RBM	0.260	4.5×10^4	3.3	0.280	5.8×10^4	3.7
BSSC	0.280	2.2×10^3	11.2	0.293	2.6×10^3	13.4
PCAH	0.252	6.5	1.2	0.235	7.4	1.9
SpH	0.275	25.8	28.3	0.284	88.2	101.9
AGH	0.161	144.7	55.7	0.267	184.2	72.0
ITQ	0.320	1.0×10^3	32.1	0.360	1.1×10^3	35.7
STH	0.270	1.2×10^3	17.4	0.318	1.8×10^3	19.8
CH	0.280	93.4	53.5	0.330	98.2	54.4
LSPH	**0.340**	1.1×10^3	20.3	**0.376**	1.2×10^3	22.7

Table 3 The comparison of MAP, training time and test time of 32 bits and 48 bits of all the compared algorithms on the GIST 1M dataset

Methods	GIST 1M					
	32 bits			48 bits		
	MAP	Train time (s)	Test time (μs)	MAP	Train time (s)	Test time (μs)
LSH	0.107	1.4	2.7	0.135	2.1	3.0
KLSH	0.110	29.5	27.2	0.120	30.7	38.0
RBM	0.123	5.5×10^4	3.4	0.142	6.2×10^4	3.7
BSSC	0.112	3.2×10^3	13.0	0.130	3.8×10^3	15.1
PCAH	0.090	49.2	2.8	0.075	52.3	3.0
SpH	0.130	65.3	40.2	0.148	131.1	116.3
AGH	0.124	242.5	83.7	0.160	279.4	95.6
ITQ	0.170	1.2×10^3	33.8	0.200	1.5×10^3	36.2
STH	0.123	1.9×10^3	21.3	0.171	2.5×10^3	25.2μ
CH	0.160	194	64.1	0.190	210.5	71.5
LSPH	**0.185**	1.4×10^3	21.8	**0.211**	1.7×10^3	24.1

Fig. 6 Some example images from the TinyImage dataset (Color figure online)

station with a 12-core 3.2GHz CPU and 120GB of RAM running the Linux OS.

6.1 Evaluation on LSPH

In this subsection, we first apply the proposed single-layer LSPH algorithm method for large-scale similarity search tasks. Two realistic datasets are used to evaluate all methods: **SIFT 1M**: it contains one million 128-dim local SIFT feature vectors (Lowe 2004). **GIST 1M**: it contains one million 960-dim global GIST feature vectors (Oliva and Torralba 2001). The two datasets are publicly available.[3]

For both SIFT 1M and GIST 1M, we respectively take $10K$ images as the queries by random selection and use the remaining to form the gallery database. To construct the training set, 200, 000 samples from the gallery database are randomly selected for all of the compared methods. Additionally, we also randomly choose another 50, 000 data samples as a cross-validation set for parameter tuning. In the querying phase, the returned points are regarded as true neighbors if they lie in the top 2 percentile points closest to a query for both two datasets. Since the Hamming lookup table is fast with hash codes, we will use the Hamming lookup table to measure our retrieval tasks. We evaluate the retrieval results in terms

[3] http://corpus-texmex.irisa.fr.

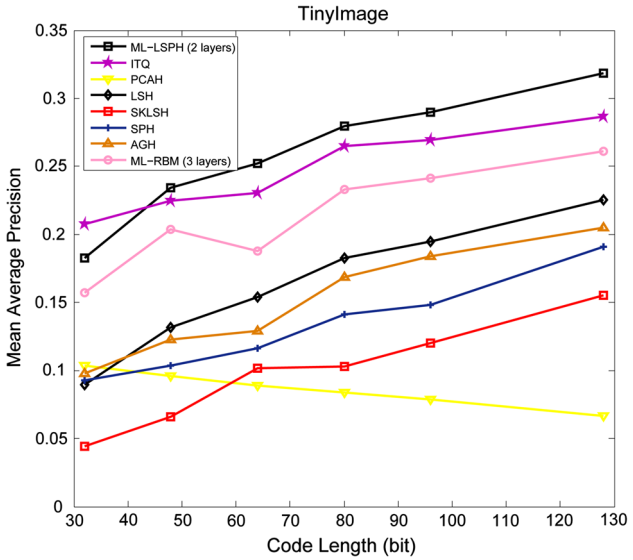

Fig. 7 The top-500 Mean Average Precision of the compared algorithms on the TinyImage dataset (Color figure online)

of the Mean Average Precision[4] (MAP) and the precision-recall curves. Additionally, we also report the training time and the test time (the average searching time used for each query) for all methods.

6.1.1 The Selected Methods and Settings

In this experiment, we compare LSPH with the 10 selected popular hashing methods including LSH (Gionis et al. 1999), BSSC (Shakhnarovich 2005), RBM (Salakhutdinov and Hinton (2007)), SpH (Weiss et al. 2009), STH (Zhang et al. 2010), AGH (Liu et al. 2011), ITQ (Gong et al. 2013), KLSH (Kulis and Grauman 2009), PCAH (Wang et al. 2012) and CH (Lin et al. 2013). In these methods, for BSSC, through the labeled pairs scheme in the boosting framework, it can obtain weights and thresholds for every hash function. RBM will be trained with several 100-100 hidden layers without fine-tuning. According to KLSH, 500 training samples and the RBF-kernel are used to output the empirical kernel map, in which we always set the scalar parameter σ to an appropriate value on each dataset. For ITQ, the number of the iterations is set as 50. In AGH with two-layer, we consider the number of the anchor points k as 200 and the number of the nearest anchors s in sparse coding as 50. CH has the same anchor-based sparse coding setting with AGH. All of the 10 methods are evaluated on different lengths of the codes, e.g., 16, 32, 48, 64, 80 and 96. Under the same experimental setting, all the parameters used in the compared methods have been strictly chosen according to their original papers.

[4] The ground-truth is defined by top 2 percentile nearest neighbors of a query via linear scan based on the Euclidean distance.

In the experiments of our LSPH method, all the data are first normalized into the range of [0, 1], since the nonnegative constraint is required in our framework. We also use the validation set to tune our hyper-parameters. Particularly, for each dataset, we select one value from $\{0.01, 0.02, \cdots, 0.10\}$ as the optimal learning rate α of multi-variable logistic regression through 10-fold cross-validation on the validation set. The regularization parameter λ is determined from one of $\{10^{-2}, 10^{-1}, 1, 10^{1}, 10^{2}, 10^{3}\}$, which yields the best performance via 10-fold cross-validation on the validation set. The regularization parameter δ in the hash function generation is fixed as $\delta = 0.35$. We limit the maximum number of iterations with 1000 in LSPH learning phase, as well.

6.1.2 Results Comparison

We demonstrate MAP curves on the SIFT 1M and GIST 1M datasets compared with different methods in Fig. 4. From the general tendency, the accuracies on the GIST 1M dataset are lower than those on the SIFT 1M dataset, since features in the GIST 1M has higher dimensions with larger variations than those on SIFT 1M. In detail of Fig. 4, ITQ always achieves higher Mean Average Precision (MAP) and gets a consistent increasing condition with the change of the code length on both datasets. Furthermore, MAP of CH also proves to be competitive but a little lower than ITQ. Interestingly, on both the SIFT 1M and GIST 1M datasets, the MAP of SpH and STH are always "rise-then-fall" when the length of code increases. Due to the use of random projection, LSH and KLSH have a low MAP when the code length is short. Moreover, PCAH always gets decreasing accuracies when the code length increases. For our method LSPH, it achieves the highest performance among all the compared methods on both datasets. The proposed LSPH algorithm can automatically exploit the latent structure of the original data and simultaneously preserve the consistency of distribution between the original data and the reduced representations. The above properties of LSPH allow it to achieve better performance in large-scale visual retrieval tasks. Fig. 5 also shows the precision-recall curves with the code length of 48 bits on both SIFT 1M and GIST 1M datasets with the top 2 percentile nearest neighbors as the ground truth. From all these figures, we can further discover that, for both two datasets, LSPH achieves apparently better performance than other hashing methods by comparing the Mean Average Precision (MAP) and Area Under the Curve (AUC). Additionally, to further illustrate the necessity of using logistic regression binarization rather than direct median value binarization as mentioned in Sect. 3.3, we carry out comparison experiments on both SIFT 1M and GIST 1M datasets. In Table 1, it is easy to observe that logistic regression binarization can achieve consistently higher MAP across all code lengthes than the direct median value binarization scheme.

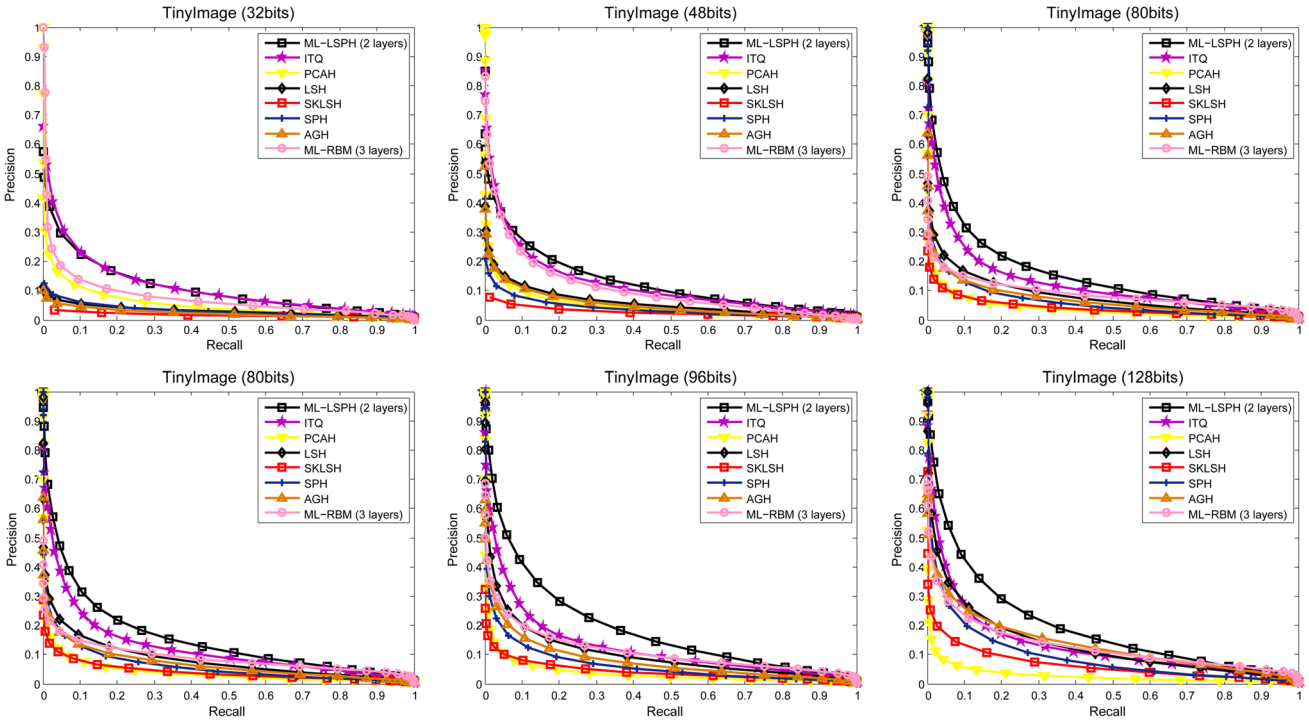

Fig. 8 Comparison of precision recall curves with different bits on the TinyImage dataset. Ground truth is defined by top-500 Euclidean neighbors (Color figure online)

Table 4 The comparison of MAP between using LSPH and ML-LSPH with different numbers of bits on all three datasets

Code length	32 bits	48 bits	64 bits	80 bits	96 bits	128 bits
SIFT 1M						
LSPH	0.340	0.376	0.424	0.445	0.461	0.470
ML-LSPH	0.347	0.382	0.441	0.470	0.492	0.498
GIST 1M						
LSPH	0.185	0.211	0.234	0.241	0.255	0.261
ML-LSPH	0.183	0.222	0.253	0.269	0.273	0.279
TinyImage						
LSPH	0.180	0.224	0.241	0.265	0.274	0.304
ML-LSPH	0.182	0.235	0.252	0.279	0.290	0.318

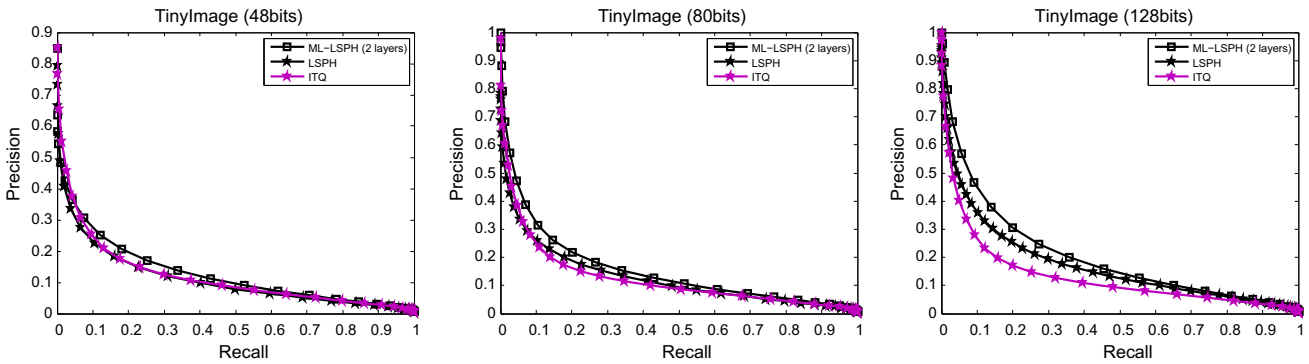

Fig. 9 Comparison of precision recall curves using ordinary LSPH (one layer) and ML-LSPH (two layers) with different bits on the TinyImage dataset (Color figure online)

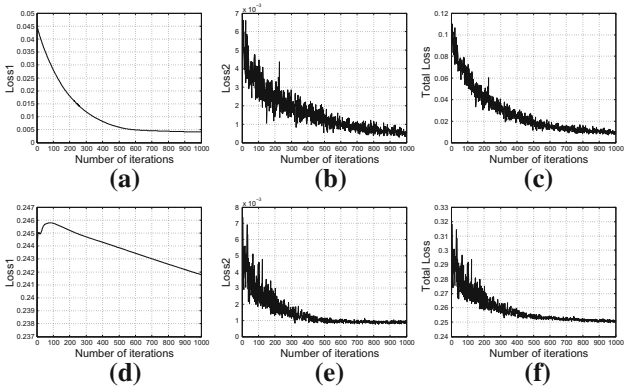

Fig. 10 Illustration of convergence on the TinyImage dataset with the code length 64 ($\lambda = 10$). For ordinary LSPH (one layer): **a** $loss1 = \|X - U_1V_1\|^2$ versus number of iterations. **b** $loss2 = KL(P||Q)$ versus number of iterations. **c** $Totalloss = \|X - U_1V_1\|^2 + \lambda KL(P||Q)$ versus number of iterations. For ML-LSPH: **d** $loss1 = \|X - g(U_1g(U_2 \cdots g(U_nV_n)))\|^2$ versus number of iterations. **e** $loss2 = \sum_{i=1}^{n} KL(P||Q^{(i)})$ versus number of iterations. **f** $Totalloss = \|X - g(U_1g(U_2 \cdots g(U_nV_n)))\|^2 + \lambda \sum_{i=1}^{n} KL(P||Q^{(i)})$ versus number of iterations. Zoom in for better viewing

Meanwhile, the training and test time for all the methods are listed in Tables 2 and 3, as well. Considering the training time, the random projection based algorithms are relatively more efficient, especially the LSH. While, RBM takes the most time for training, since it is based on a time-consuming deep learning method. Our method LSPH is significantly more efficient than STH, BSSC and RBM, but slightly slower than ITQ, AGH and SpH. It is noteworthy that once the optimal hashing functions of our method are obtained from the training phase, the optimized hashing functions will be fixed and directly used for new data. In addition, with the rapid development of silicon technologies, future computers will be much faster and even the training will become less a problem. In terms of the test phase, LSH is the most efficient methods as well, since only a simple matrix multiplication and a thresholding are needed to obtain the binary codes. AGH and SpH always take more time for the test phase. Our LSPH has the competitive efficiency with STH. Therefore, in general, it can be concluded that LSPH is an effective and relatively efficient method for the large-scale retrieval tasks.

6.2 Evaluation on ML-LSPH

In this subsection, the multi-layer LSPH is evaluated on the **TinyImage** dataset, which is a subset containing 500,000 images[5] from ***80 Million Tiny Images (Torralba et al. 2008a, b). Some example images from the TinyImage dataset are illustrated in Fig. 6. We further take $1K$ images as the queries by random selection and use the remaining to form the gallery database. Considering the cost of computation in multi-layer networks, in this experiment, only 100, 000 randomly selected samples from the gallery database form the training set. Similar to the experiments of LSPH, another 50, 000 data samples are also randomly chosen as a cross-validation set for parameter tuning. For image searching tasks, given an image, we describe it with 512-dimensional GIST descriptors (Oliva and Torralba 2001) in this experiment and then learn to hash these descriptors with all compared methods. In the querying phase, a returned point is regarded as a neighbor if it lies in the top ranked 500 points for the TinyImage dataset. We evaluate the retrieval results through Hamming distance ranking and report the Mean Average Precision (MAP) and the precision-recall curves by changing the number of top ranked points. Additionally, we also report the parameter sensitive analysis and visualize some retrieved images of compared methods on this dataset.

To avoid confusion, in this experiment, we only compare with LSH, PCAH, ITQ, AGH, RBM and SpH, which have shown distinctive performance according to the results in the previous comparison. Besides, we also add a new hashing technique SKLSH in this experiment. Note that, RBM here has 100-100-100 three hidden layers without fine-tuning. All of the above methods are then evaluated on six different lengths of codes (32, 48, 64, 80, 96, 128). Under the same experimental setting, all the parameters used in the compared methods have been strictly chosen according to their original papers.

For our ML-LSPH, the data zero-one normalization and hyper-parameter selection follow the same scheme as those in LSPH. Besides, to better reach the local minimum loss in ML-LSPH, the learning rate γ for iterative optimization is considered. In this experiment, we fix $\gamma = 0.5$ for ML-LSPH. More importantly, for the reason mentioned above that when the number of the layers increases, the accumulative reconstruction error will cause the non-convergence problem of the proposed model (Trigeorgis et al. 2014), we evaluate ML-LSPH with $n = 2$ layers, which is the same setting as in Trigeorgis et al. (2014). We further set the dimension of the middle layer[6] (i.e., V_1) to 256 on this dataset.

[5] This subset is downloaded from http://groups.csail.mit.edu/vision/TinyImages/.

[6] After attempting various network architectures with different dimensions of middle layers, the current ML-LSPH network is the best option for our task. Similar to the deep learning model (Krizhevsky et al. 2012; Szegedy et al. 2014). It is noteworthy that the dimensions of middle layers are quite sensitive to the data distribution (i.e., dataset) and there is no particular proof to explain what length of dimension for middle layers is better. Thus, we recommend to try different structure settings in order to determine what kind of structure can achieve the best performance.

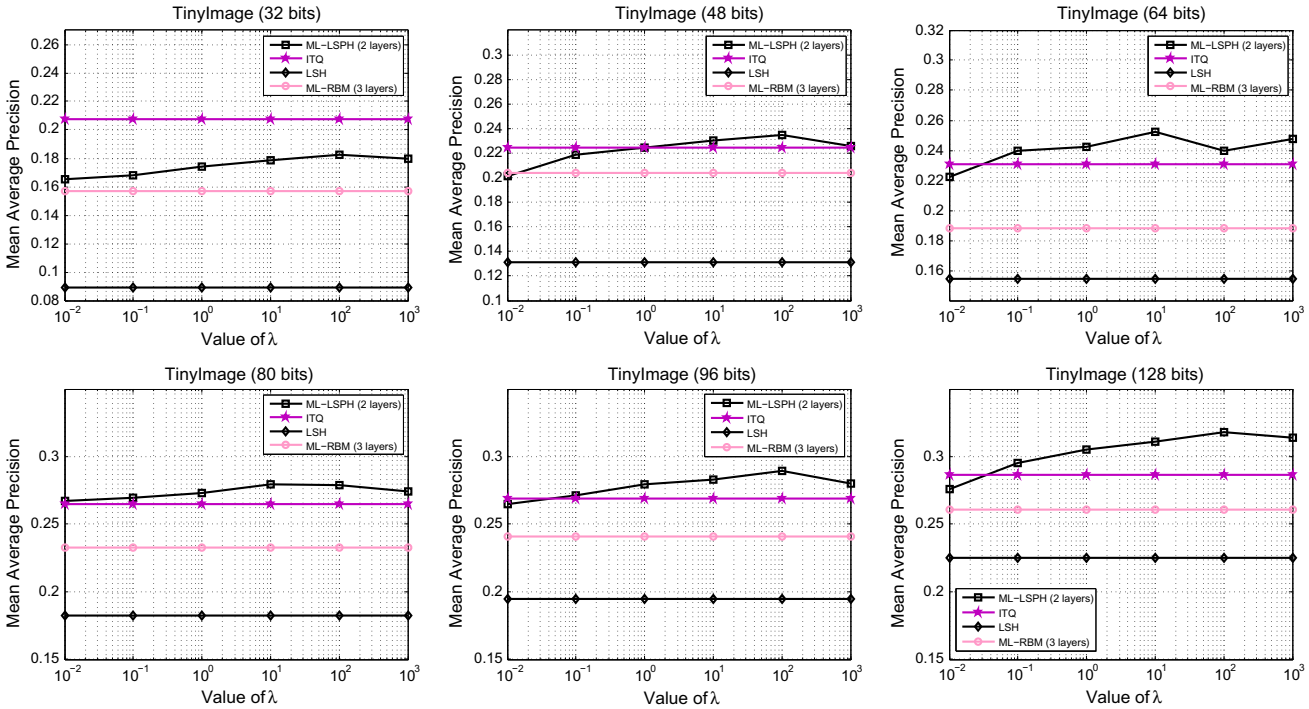

Fig. 11 Parameter sensitivity analysis of the regularization parameter λ with different bits on the TinyImage dataset (Color figure online)

Table 5 Results comparison (MAP) of ML-LSPH with/without KL regularization term $\sum_{i=1}^{n} KL(P||Q^{(i)})$

Code length	32 bits	48 bits	64 bits	80 bits	96 bits	128 bits
With KL regularization term ($\lambda = 1$)	0.174	0.224	0.242	0.273	0.280	0.305
Without KL regularization term ($\lambda = 0$)	0.170	0.195	0.218	0.246	0.254	0.273

6.2.1 Results Comparison

Fig. 7 illustrates the MAP curves of all compared algorithms on the TinyImage dataset. Our ML-LSPH algorithm achieves slightly lower MAP than ITQ when the code length is less than 48 bits but consistently outperforms all other compared methods in every length of code. RBM with 3 layers can also produce competitive search accuracies on this dataset. Different to other hashing techniques, the performance of PCAH decreases with the increase of the code length. The similar phenomenon has appeared in the previous evaluation on SIFT 1M and GIST 1M datasets. The performance of AGH, SpH and LSH is consistent with that in the previous experiments. Besides, Fig. 8 shows a series of precision-recall curves with different code lengths on the TinyImage dataset with the 500 nearest neighbors as the ground truth. By comparing the Area Under the Curve (AUC), our ML-LSPH achieves apparently better performance than other methods on relatively long bits (code length $\geq$ 48 bits) and the performance slightly goes down when short hash codes are adopted. Moreover, in Table 4 and Fig. 9, we also compare the performance between ML-LSPH and LSPH in terms of the MAP and AUC on all three datasets. The ML-LSPH can achieve consistently better results than LSPH, since large intra-class variations in TinyImage cause complex and noisy data distributions, which are more difficult to handle by LSPH. Besides, Fig. 10 illustrates the convergence of the proposed LSPH and ML-LSPH on TinyImage with the code length of 64. We can clearly observe that, for LSPH, the loss of $\|X - U_1 V_1\|^2$ dramatically drops down when the number of iterations increases. While, for ML-LSPH, the loss of $\|X - g(U_1 g(U_2 \cdots g(U_n V_n)))\|^2$ first climbs up when the number of iterations is less than 50 and then goes down. With the batch-based learning scheme, the total losses of both LSPH and ML-LSPH can steadily decrease with little fluctuation.

Additionally, in Fig. 11, we also compare the retrieval performance of ML-LSPH with respect to the regularization parameter λ along different code lengths via cross-validation. When tuning the parameter λ from 0.01 to 1000 with a scale factor of 10, the MAP curves of ML-LSPH appear to be relatively stable and insensitive to λ. For code lengths equal to 64 bits and 80 bits, the best performance occurs when $\lambda = 10$. However, for the rest of the code lengths, ML-LSPH can achieve the highest retrieval accuracies with $\lambda = 100$.

Fig. 12 The top 25 retrieved images for queries (plane, bird, car, horse, ship and truck) with 96 bits using ML-RBM, LSH, SpH, PCAH, SKLSH, ITQ, AGH and our ML-LSPH (from **a** to **h**). Best viewed in color (Color figure online)

Specifically, to illustrate the effectiveness of the data distribution preserving (KL divergence) regularization term in ML-LSPH, we also compare the algorithm without using the KL divergence term in Table 5. The results indicate that combining the multi-layer NMF network with the data distribution preserving term could always gain better performance. Finally, the top-ranked retrieval results using compared methods on some typical queries are illustrated in Fig. 12. It can be observed that the retrieved images via ML-LSPH have more semantic consistency with the query images.

7 Conclusion and Future Work

In this paper, we have proposed the Latent Structure Preserving Hashing (LSPH) algorithm, which can find a well-structured low-dimensional data representation through the Nonnegative Matrix Factorization (NMF) with the probabilistic structure preserving regularization part, and then the multi-variable logistic regression is effectively applied to generate the final hash codes. To better tackle the data with complex and noisy data distributions, a multi-layer LSPH (ML-LSPH) extension has also been developed in this paper. Extensive experiments on three large-scale datasets have demonstrated that our algorithms can lead to competitive hashing performance for large-scale retrieval tasks.

As we mentioned in the introduction of the paper, the hash codes learned from both LSPH and ML-LSPH can be regarded as independent latent attributes due to the property of NMF. In future work, this kind of learned data-driven attributes will be further explored with zero-shot learning for unseen image classification, retrieval and annotation tasks.

References

Ahn, J. H., Choi, S., & Oh, J. H. (2004). A multiplicative up-propagation algorithm. In *International Conference on Machine Learning*. ACM, New York.

Baluja, S., & Covell, M. (2008). Learning to hash: Forgiving hash functions and applications. *Data Mining and Knowledge Discovery*, *17*(3), 402–430.

Bian, W., & Tao, D. (2010). Biased discriminant euclidean embedding for content-based image retrieval. *IEEE Transactions on Image Processing*, *19*(2), 545–554.

Cai, D., He, X., & Han, J. (2007). Spectral regression for efficient regularized subspace learning. In *International Conference on Computer Vision*.

Cai, D., He, X., Han, J., & Huang, T. S. (2011). Graph regularized nonnegative matrix factorization for data representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(8), 1548–1560.

Cai, Z., Liu, L., Yu, M., & Shao, L. (2015). Latent structure preserving hashing. In *British Machine Vision Conference*.

Cao, L., Li, Z., Mu, Y., & Chang, S. F. (2012). Submodular video hashing: a unified framework towards video pooling and indexing. In *Proceedings of the ACM international conference on Multimedia*, pp. 299–308.

Gao, Y., Shi, M., Tao, D., & Xu, C. (2015). Database saliency for fast image retrieval. *IEEE Transactions on Multimedia*, *17*(3), 359–369.

Gionis, A., Indyk, P., & Motwani, R. (1999). Similarity search in high dimensions via hashing. *Very Large Date Bases*, *99*, 518–529.

Gong, Y., Lazebnik, S., Gordo, A., & Perronnin, F. (2013). 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.

Gu, Q., & Zhou, J. (2009). Neighborhood preserving nonnegative matrix factorization. In *British Machine Vision Conference*.

Guan, N., Zhang, X., Luo, Z., Tao, D., & Yang, X. (2013). Discriminant projective non-negative matrix factorization. *PLoS One*, *8*(12), e83291.

Heo, J. P., Lee, Y., He, J., Chang, S. F., & Yoon, S. E. (2012). Spherical hashing. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*(7), 1527–1554.

Hosmer, D. W, Jr., & Lemeshow, S. (2004). *Applied logistic regression*. New York: Wiley.

Hoyer, P. O. (2004). Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, *5*, 1457–1469.

Jayaraman, D., & Grauman, K. (2014). Zero-shot recognition with unreliable attributes. *Advances in Neural Information Processing Systems*, *4*, 3464–3472.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*.

Kulis, B., & Grauman, K. (2009). Kernelized locality-sensitive hashing for scalable image search. In *International Conference on Computer Vision*.

Lampert, C. H., Nickisch, H., & Harmeling, S. (2014). Attribute-based classification for zero-shot visual object categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(3), 453–465.

Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, *401*(6755), 788–791.

Lee, D. D., & Seung, H. S. (2000). Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing System*.

Li, P., Bu, J., Yang, Y., Ji, R., Chen, C., & Cai, D. (2014). Discriminative orthogonal nonnegative matrix factorization with flexibility for data representation. *Expert Systems with Applications*, *41*(4), 1283–1293.

Li, S.Z., Hou, X., Zhang, H., & Cheng, Q. (2001). Learning spatially localized, parts-based representation. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Lin, Y., Jin, R., Cai, D., Yan, S., & Li, X. (2013). Compressed hashing. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Liu, L., & Shao, L. (2015). Sequential compact code learning for unsupervised image hashing. *IEEE Transactions on Neural Networks and Learning Systems*.

Liu, L., Yu, M., & Shao, L. (2015). Multiview alignment hashing for efficient image search. *IEEE Transactions on Image Processing*, *24*(3), 956–966.

Liu, L., Yu, M., & Shao, L. (2015). Projection bank: From high-dimensional data to medium-length binary codes. In *International Conference on Computer Vision*.

Liu, L., Yu, M., & Shao, L. (2015). Unsupervised local feature hashing for image similarity search. IEEE Transactions on Cybernetics.

Liu, W., Wang, J., Ji, R., Jiang, Y. G., & Chang, S.F. (2012). Supervised hashing with kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2074–2081.

Liu, W., Wang, J., Kumar, S., & Chang, S. F. (2011). Hashing with graphs. In *International Conference on Machine Learning*.

Liu, Y., Wu, F., Yang, Y., Zhuang, Y., & Hauptmann, A. G. (2012). Spline regression hashing for fast image search. *IEEE Transactions on Image Processing*, *21*(10), 4480–4491.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, *60*(2), 91–110.

Masci, J., Meier, U., Cireşan, D., & Schmidhuber, J. (2011). Stacked convolutional auto-encoders for hierarchical feature extraction. In *Artificial Neural Networks and Machine Learning* (pp. 52–59).

Oliva, A., & 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.

Qin, J., Liu, L., Yu, M., Wang, Y., & Shao, L. (2015). Fast action retrieval from videos via feature disaggregation. In *British Machine Vision Conference*.

Raginsky, M., & Lazebnik, S. (2009). Locality-sensitive binary codes from shift-invariant kernels. In *Advances in Neural Information Processing Systems*.

Salakhutdinov, R., & Hinton, G. (2009). Semantic hashing. *International Journal of Approximate Reasoning*, *50*(7), 969–978.

Salakhutdinov, R., & Hinton, G. E. (2007). Learning a nonlinear embedding by preserving class neighbourhood structure. In *International Conference on Artificial Intelligence and Statistics*.

Schönemann, P. H. (1966). A generalized solution of the orthogonal procrustes problem. *Psychometrika*, *31*(1), 1–10.

Shakhnarovich, G. (2005). Learning task-specific similarity. Ph.D. thesis, Massachusetts Institute of Technology.

Song, J., Yang, Y., Huang, Z., Shen, H. T., & Luo, J. (2013). Effective multiple feature hashing for large-scale near-duplicate video retrieval. *IEEE Transactions on Multimedia*, *15*(8), 1997–2008.

Song, J., Yang, Y., Li, X., Huang, Z., & Yang, Y. (2014). Robust hashing with local models for approximate similarity search. *IEEE Transactions on Cybernetics*, *44*(7), 1225–1236.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Tao, R., Smeulders, A. W., & Chang, S. F. (2015). Attributes and categories for generic instance search from one example. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Torralba, A., Fergus, R., & Freeman, W. T. (2008). 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *30*(11), 1958–1970.

Torralba, A., Fergus, R., & Weiss, Y. (2008). Small codes and large image databases for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Trigeorgis, G., Bousmalis, K., Zafeiriou, S., & Schuller, B. (2014). A deep semi-nmf model for learning hidden representations. In *International Conference on Machine Learning*.

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(11), 2579–2605.

Wang, D., Gao, X., & Wang, X. (2015). Semi-supervised constraints preserving hashing. *Neurocomputing*, *167*, 230–242.

Wang, D., Gao, X., Wang, X., & He, L. (2015). Semantic topic multimodal hashing for cross-media retrieval. In *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 3890–3896).

Wang, J., Kumar, S., & Chang, S. F. (2012). Semi-supervised hashing for large-scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *34*(12), 2393–2406.

Wang, Q., Zhu, G., & Yuan, Y. (2014). Statistical quantization for similarity search. *Computer Vision and Image Understanding*, *124*, 22–30.

Weiss, Y., Torralba, A., & Fergus, R. (2009). Spectral hashing. In *Advances in Neural Information Processing Systems*.

Xie, B., Mu, Y., Tao, D., & Huang, K. (2011). m-sne: Multiview stochastic neighbor embedding. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *41*(4), 1088–1096.

Yu, F. X., Cao, L., Feris, R. S., Smith, J. R., & Chang, S. F. (2013). Designing category-level attributes for discriminative visual recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Yu, F.X., Kumar, S., Gong, Y., & Chang, S.F. (2014). Circulant binary embedding. arXiv preprint arXiv:1405.3162.

Yu, J., Tao, D., Wang, M., & Rui, Y. (2015). Learning to rank using user clicks and visual features for image retrieval. *IEEE Transactions on Cybernetics*, *45*(4), 767–779.

Yuan, Z., & Oja, E. (2005). Projective nonnegative matrix factorization for image compression and feature extraction. Image Analysis (pp. 333–342).

Zhang, D., Wang, J., Cai, D., & Lu, J. (2010). Self-taught hashing for fast similarity search. In *Conference on Special Interest Group on Information Retrieval*.

Zhang, D., Zhou, Z.H., & Chen, S. (2006). Non-negative matrix factorization on kernels. In *Pacific Rim International Conference on Artificial Intelligence*.

Zhang, X., Yu, F.X., Guo, R., Kumar, S., Wang, S., & Chang, S.F. (2015). Fast orthogonal projection based on kronecker product. In *International Conference on Computer Vision*.

Zheng, W., Qian, Y., & Tang, H. (2011). Dimensionality reduction with category information fusion and non-negative matrix factorization for text categorization. In *Artificial Intelligence and Computational Intelligence*.